Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Projecte Fi de Carrera

New Hardware Architecture for Low-Cost Functional Test Systems. Application to HDMI Generation

Marc Blanch Elias

Estudis: **Enginyeria de Telecomunicació**
Director: **Jordi Madrenas Boadas**
Any: **Juliol 2011**

Al meu Pare

Agraïments / Acknowledgements

A Cori, companya de viatge, no se't dona bé el mapa de registres, però en aguantar-me a mi i a les meves plaques ets una crack. Aquest viatge, m'hauria agradat que fos més curt, però també està fet amb tu. Ara a preparar-nos per el següent.

A la meva família per donar-me tant de suport incondicional, ara toca disfrutar-ho un moment.
A la meva Mare que sempre dona exemple, gràcies per estimar-me tant, encara que faci moltes coses malament i sigui de poca disciplina, m'agrada pensar que el Pare estaria orgullós de mi en aquest moment i ho disfrutaria com el que més.
A Tete per les seves abraçades d'ós i les merescudes mini-bronques, al no abusar, les poques rebudes les recordo i em van fer obrir els ulls.
A Viqui per l'exemple de superació, tots sou uns currantes.

A tots els magnífics companys del grup de CBA per la seva relació especial amb aquest projecte, nosaltres vam aixecar-lo i vam tenir el valor de demostrar col·lectivament la nostra capacitat professional davant d'ulls que no volen veure. Són moltes les coses que m'heu ensenyat per recórrer el camí i que haig d'agraïr-vos:
A Esteve, gràcies per fer-me pensar sempre un punt més enllà, ser crític i no deixar que m'acomodi.
A Salva por aguantarme siempre y darme algo de tu templanza, me quedo con poca pero mejor que nada.
A Jaume, una de las piedras angulares, gracias por unas buenas líneas diferenciales y algún rework. Tranquilo que la Ethernet Proto funcionaría bien.
A Sergio també per aguantar-me i donar-me exemple diari de practicitat.
A Oscar per posar una orella i voler aprendre, no tothom vol i s'agraeix.
I finalment a Laura, companya de projectistes anónims, per liderar donant exemple i ensenyar-me el bon camí, sense la teva llum no veia mai el final del túnel.

No em vull oblidar dels més ex- de tots, Alex per estar quan cal, no saps el que se't troba a falta en el dia a dia, a Josep Maria, tu no te'n adones però lo que arribes a ensenyar, a mi m'has fet enginyer, i a Javi, per els teus consells o idees que sempre fan pensar.

Łukasz, eventually we are going to visit you as soon as possible.

A Andrea, Josep, per obrir-me les portes de casa seva de bat a bat, i a tants altres que no estàn en aquestes línies però que s'alegren per mi de que arribi a aquest moment, ja sé que hauria hagut de posar-me abans, pero ara ja està.

Finalment, una menció especial al meu tutor Jordi, per fer aquest esforç més enllà del que t'era estrictament necessari. Ho agraeixo molt.

Abstract / Resum del projecte

<English version>
The current bad economic context forces test engineering teams to find new imaginative test solutions to keep the inspection and test standards while reducing the level of investment in specialized test equipment. In this context, this work discusses a new hardware architecture proposal for the functional test systems present in BCN Tec factory. The new proposed architecture is focused on flexibility, to give quick test solutions to any electronic board that introduces new test challenges, while keeping the pace in terms of performance against much more expensive solutions. The proposed architecture is compared with the current one in BCN Tec automated functional test systems, based on the traditional approach of commercial test equipment.
This work consists of two well-defined parts, a review of the several aspects related with inspection and test of PCBs as well as the presentation of the general concepts of the new hardware architecture, followed by a demonstration of the proposed low-cost strategy by means of a proof-of-concept development.
The implementation in a new hardware platform is focused from the start in the scalability and design reuse to minimize the non-recurring engineering (NRE) costs, in order to quickly react proposing a new feature for the "platform" when a new technology has to be introduced. The work continues with the proof-of-concept prototype of the new hardware architecture, that challenges one of the "premium" features of commercial test equipment for TV: HDMI generation.
The proof-of-concept prototype development phases are introduced linearly following a development flow that later can be used as a pattern to introduce new technologies in the final platform. Emphasis is placed on the platform design approach, and these parts of the development are extensively discussed and separated from the parts exclusively developed for the HDMI generation.
After the embedded development part, containing system, hardware and software development is finished, the work focuses on the power up & verification of the HDMI generator board, and the development of host computer software to test the proof-of-concept prototype in production. With all the results of the different parts of development collected, this thesis ends with the conclusions and hints of the future lines of work and research.

<Versió en català>
L'actual mal moment econòmic porta als grups d'enginyeria a esforçar-se per trobar noves solucions imaginatives per mantenir els estàndards de test i inspecció, en un context de reducció quasi absoluta d'inversions en equips especialitzats de test.
En aquest entorn, el treball desenvolupa una nova arquitectura hardware per als equips automàtics de test funcional de la fàbrica de BCN Tec. La nova arquitectura proposada està centrada en aconseguir molta flexibilitat, per tal de donar solucions de test econòmicament viables per a qualsevol circuit electrònic que introdueixi una nova tecnologia a testejar en producció, mantenint però el nivell de qualitat del test comparat amb el que et pugui oferir una solució comercial molt més cara. L'arquitectura de hardware de test proposada es compara amb l'actual en les màquines automàtiques de test funcional de la fàbrica, basades en equipament de test comercial.
En aquest sentit el treball s'estructura en dos parts ben definides, la primera una revisió dels aspectes generals relacionats amb l'inspecció i test de PCBs així com la presentació de les línies generals de la nova arquitectura hardware, i una segona amb una demostració de la

estrategia de hardware de baix cost proposada, per mitjà del desenvolupament d'un prototip de prova de concepte.
La implementació d'aquesta arquitectura en una plataforma hardware agafa com a premisses l'escalabilitat i la reutilització, per minimitzar els costs fixes de desenvolupament i tenir una capacitat de reacció ràpida a qualsevol nova necessitat de test que s'hagi d'introduir i que es pugui afegir a la plataforma hardware.
El treball continua amb el prototip de prova de concepte de la nova arquitectura hardware que s'agafa com a repte una de les característiques "premium" de molts equips comercials de test per a TV: la generació de HDMI.
Les fases de desenvolupament del prototip de prova de concepte estan organitzades de forma lineal, seguint un flux de desenvolupament que pot ser utilitzat després com a patró per afegir noves tecnologies a la plataforma hardware final. Es posa especial èmfasi en el concepte de plataforma de hardware oberta, i les parts de desenvolupament de la plataforma estan extensament discutides i separades de les parts exclusivament desenvolupades per a la generació de HDMI.
Després de les parts del desenvolupament del sistema embedded, tant de planificació del sistema, com hardware específic i firmware, el treball passa a centrar-se en l'arrencada i verificació del prototip desenvolupat, la placa de generació d'HDMI. Després passa a explicar el desenvolupament del software de control des del PC, per tal de testejar el funcionament del prototip a producció de forma automàtica. Els resultats aconseguits en les diferents parts del desenvolupament es presenten en les conclusions, i s'introdueixen les feines relacionades per continuar amb el projecte així com futures línies de desenvolupament.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scenario

The purpose of this project is to develop a new low cost HW architecture for functional test inspection automatic machines at board-level and demonstrate the essential concepts of new architecture in a proof-of-concept prototype with basic functionalities. The new architecture is going to be flexible and adaptable enough to be ready not only for the usual Sony LCD TV PCB boards produced in BCN Tec, but also to offer test services at a low implementation cost for any other electronic board assembled in BCN Tec, as part of his electronic Contract Manufacturing (ECM/CMD) business activities.

In this context, the new direction of the BCN Tec facility, which was sold by Sony to Ficosa, accelerated the already started reconversion to a more diversified factory, especially for the AutoMount area, that started to produce other electronic products instead of their typical production of LCD TV main boards.

1.2 Motivation

All the fields related with electronics industry are in constant evolution, and in a challenging period where BCN Tec factory needed to consolidate his role, some re-thinking of critical processes was needed to be better prepared for the short and mid-term scenario.

Originally, the definition phase of this project was initiated because Sony world wide (WW) directives regarding Standard Production Process for TV (STD process) were changed. Basically new directives were focusing on board-level inspection to achieve similar inspection coverage at a lower cost compared with system-level inspection of the mounted TV set. The main target of this directive was to reduce man power and stations needed in system-level inspections, substituting them with automated board-level inspections and keeping the same coverage.

In BCN Tec, board-level inspections were already automated in comparison with other factories, but in a combined factory where both PCB manufacturing (AutoMount Area) and final TV manufacturing (Final Assembly Area) are in the same location, the needed inspections to assure the desired quality in the market can be balanced between the entire production chain. That was the reason that for the last two years, the board-level inspections were only made in a sampling basis and not for all the boards produced.

As these new directives were not strictly mandatory, and because of high-level of automation for system-level inspections already present in BCN Tec, the application in the factory was not deemed as necessary, unless automation for system-level inspections with the mounted TV set could not be used.

Anyway, the automation of inspections relied in an Automatic Connection Unit, that was installed in the Final Assembly (FA) Lines with the objective to automatically connect all the vertical user connectors present in the rear cover of a TV set, reducing manpower, because usually those connections were made by an operator. But Sony TV PCB boards for 2011 line-

up included one board that was targeting flat LCD TV market, with all connectors in horizontal orientation in the left and bottom side of the TV, instead of the typical vertically-oriented connectors in the back side of the TV. The ACU connection relied in the vertically-oriented connectors, so in that new kind of boards all the automations of the system-level inspection were impossible to use. In that case all the automation in the FA lines could not be used, the current functional test machines of PCB, also called CBAs (Circuit Board Adjustment), could take responsibility to keep quality levels increasing board-inspection coverage, and going back to a full test of the production instead of only a sampling.

Current BCN Tec CBA, shown in Figure 1.1, consists of two differentiated blocks

- The common mechanical pressure unit with up and down movement in the vertical axis, and a complete variety of common test equipment,
- The chassis-specific base unit, that is the mechanical fixture adaptable to any pressure unit due to a common interconnection interface, and that needs to be customized and redesigned for every new device under test (DUT).

Figure 1.1. Current CBA for functional test of LCD TV main boards

The main contribution of the current CBA was the mechanical fixture, designed to connect directly the connectors of the board instead of using the typical bed-of-nails fixtures. That very successful change of philosophy was made to extend the test coverage to the same level of system-level inspection of mounted TV sets, and to reduce the high NFF (No Failure Found) ratio of typical bed-of-nails solutions. As the vertical movement was the hallmark of the CBA, it was also adaptable by using small air cylinders to connect to boards with horizontal user connectors. In the photos shown in Figure 1.2 it can be appreciated the vertical movement together with details of the base unit and the interconnection between the two blocks detailed before.

From hardware point of view, the equipment in current CBA was selected to cover any test necessities of LCD TV main boards.

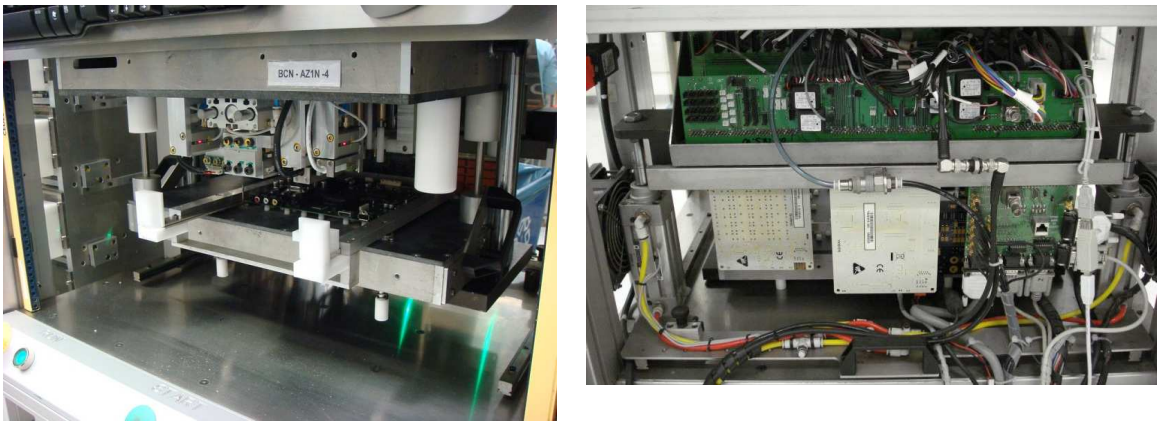


Figure 1.2. Detail of the CBA base unit and the interconnection between the two blocks

Anyway, to take over all the inspections made in the TV line for 100% of the production instead of the 20% sampling, the current park of CBAs was not enough.
There were some strong reasons that advised against new investments in the same direction, that will be very difficult to recover in a short time.

- The high cost of the common mechanical pressure unit.
- The poor usage of one of its main features, the vertical movement, in the new boards lineup
- The exclusive use of the equipment, only useful to test LCD TV main boards and not flexible enough to give test solutions for other electronic products.

In this situation, a more open, smart and flexible architecture, focused in giving quick and innovative solutions to cut down the production costs is proposed. This CBA low-cost alternative has three key aspects:

- A new **mechanical architecture**, based on a rack approach with individual test positions for boards, and with automatic connections made by small air cylinders instead of a big pneumatic fixture. That approach is more flexible to any kind of electronic boards.
- A new **hardware architecture**, based in individual homemade equipment for every test position, with an enough flexible architecture to cover all test necessities that could appear in the mid-term with quick and easy to make upgrades, and a robust control interface, appropriate for an industrial environment. That approach solves the multiplexing problems of the shared hardware and reduces the investment in equipment for test.
- A new **software architecture**, based on having only one host controller for every rack of test positions. As all the equipment and control for every test position is independent, the new architecture enables a full parallel mode with only one test controller, because the tasks of the controller have been reduced to the minimum possible.

Although they are barely explained to show a general overview to the reader, this project is not centered on the mechanical and software architectures, as some of their aspects had already been tested in other machines already present in BCN Tec production lines, the parallel In-Circuit Software Writers, shown in Figure 1.3 on the left, based on the rack

approach and with only one controller (host PC) for every rack position. So in these points, the proposed new CBA low-cost machine is only an evolution of the original concept, see also in Figure 1.3 on the right side.

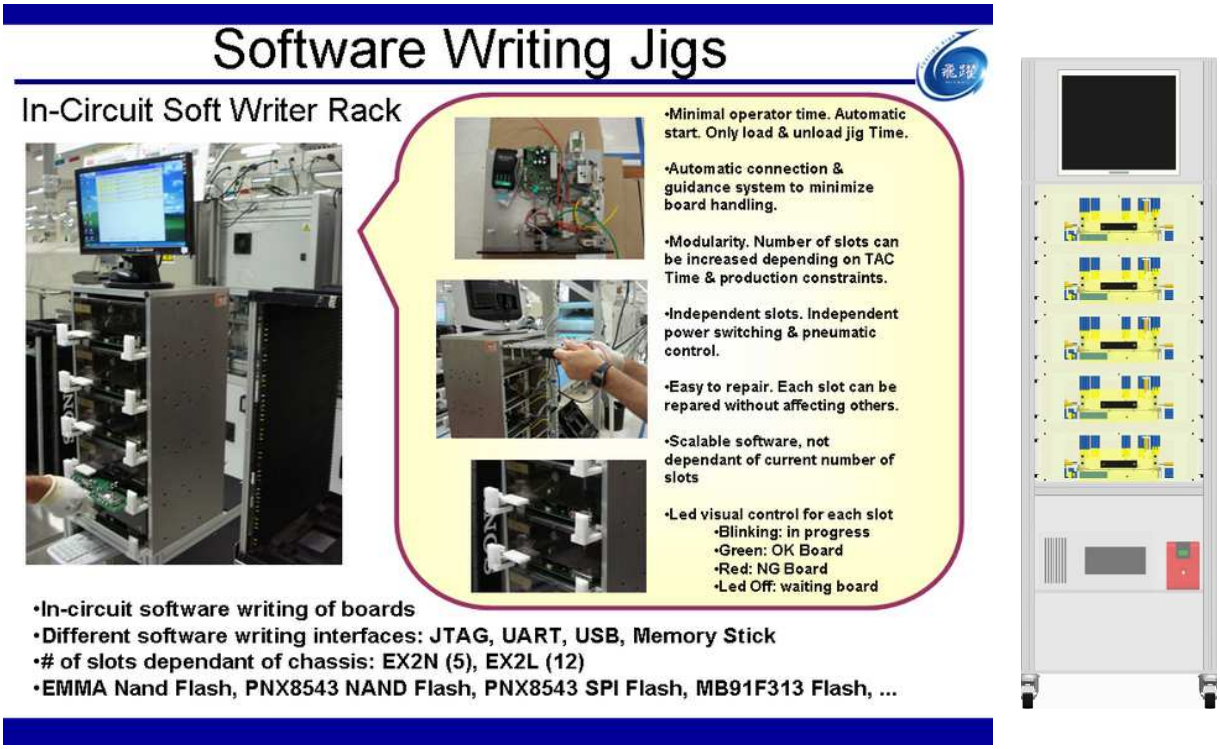


Figure 1.3. In-Circuit software writers overview (left side) and CBA Low-Cost first draft (right side)

As it have been previously exposed some lack of definition times came with the BCN Tec new owner's takeover and the first motivation of the project disappeared when the new high-end flat LCD TV models were left out of the new models to introduce in BCN during 2011.

Anyway current CBAs are already being used to test other electronic products, and the new CBA low-cost proposal was open and flexible enough to give solutions for any other electronic products, either for CMD or Automotive businesses

- Main features and general philosophy of CBA Low-Cost is completely flexible to any kind of PCB production.
- CBA Low-Cost is an open platform to develop new test features needed for future production.
- All outputs of different sub-projects can be used independently.
    - New hardware designed can be used in different processes of production chain.
    - Mechanical design concept can be re-used for different test types, not only functional test.

1.3 Objectives

The present research work pretends to explore the challenge of creating a new hardware architecture for the CBA low-cost machines, focused on developing customized homemade equipment, with modern features to be competitive with current commercial test equipment solutions for a much lower price, and with the enough flexibility to adapt the hardware to any new test challenge in the near future with as minimum effort as possible. Although the new

hardware architecture is only a part of the whole CBA Low-Cost project, its outputs are completely reusable in any other customized test solution, and to establish a design to reuse philosophy is also one of the main goals of this project.

The stated main objective can be divided in the following specific objectives:

- Presentation of the functional test necessities and state of the art of the functional test.
- High-level analysis of the problem and economical comparison with current solutions for functional test used in BCN Tec in Part I.
- Selection of an appropriate embedded architecture in line with the general project objectives.
- Development of a proof-of-concept prototype using a linear development flow and a methdololgy to be used as a pattern for future developments.
- Challenge a new technology in the proof-of-concept prototype: HDMI generation.
- Development of a platform system architecture reusable in future developments.
- Development of cores for the HDMI generation.
- Development of software modules to control all the cores developed.
- Development of a reusable software platform friendly enough to introduce quickly any new functionality.
- Development of automation libraries to integrate the developed hardware in production trials.

1.4 Overview

This report is divided in two main parts: hardware architecture and proof-of-concept. Each of the parts is then divided into several chapters.

Part 1 exposes the new hardware architecture proposal for functional test systems and remarks the open platform design approach.

Chapter 2 starts explaining the basic concepts of PCB manufacturing and test, and the evolution made on these fields during the last years in order to explain the main reasons behind the current BCN Tec test strategy.

Chapter 3 goes deeply to the functional test basics of the more typical but also more technologically advanced product present in BCN Tec production, the LCD TV main board, that is the benchmark to start exploring the features needed in the equipment developed for the new hardware architecture. The new hardware architecture proposal is introduced for the first time and briefly compared in economic terms with the hardware architecture of the current CBA.

Chapter 4 explores the requirements of the new hardware architecture from a high-level point of view, in order to correctly decide which embedded architecture is the more suitable to develop the new hardware platform. Once the selection is made, it explains the fundamentals of the design methodology and related tools for the selected embedded solution. This methodology flow is used throughout the development work, and the following chapters are presented also in this order.

Part 2 introduces the proof-of-concept prototype and all its development phases, so the project leaves the theoretical issues, and is oriented to the embedded system development.

Chapter 5 explains the decision to develop a proof-of-concept prototype to show project long-term feasibility and goes deeply in the functionalities that the proof-of-concept prototype is targeting. The specific details of the prototype architecture are also described, and this chapter is used as the framework to start the specific embedded development phases of the prototype. The project focuses on the proof-of-concept prototype and its development that must show the strengths of the proposal and its feasibility, in order to overcome this milestone and go ahead with the next phases of the project

Chapter 6 presents the main decisions during schematic & pcb development of the HDMI transmitter daughter board, the expansion card that gives the HDMI functionality to the embedded system.

Chapter 7 introduces the system architecture and shows how the system is designed and planned to integrate other external peripherals outside of the main NiosII system block.

Chapter 8 focuses on the strategy planned for an easy system integration of external cores, and the hardware configuration of the CycloneII device resources, like PLLs.

Chapter 9 follows on the hardware architecture development but focuses directly on the external cores integration and/or development, especially the audio and video generation cores that are essential for the HDMI generator board.

Chapter 10 explains the software development phases followed to build a functional software for the HDMI generator board platform, especially focusing on the driver development for the specific cores integrated in the hardware and the control and configuration of the ADV7511 HDMI transmitter.

Chapter 11 explains the complete system power-up and the successive phases to test all the board features. Eventually the complete system is validated to check if it fulfills the targeted features.

Chapter 12 leaves the embedded system development to introduce the generic Ethernet control libraries that has been developed to automate the board configuration, and be able to introduce the prototype in production. It also introduces a specific control application for debugging.

Chapter 13 explores the last phase of the prototype development, the trials of the product in production to assure long-term reliability

Eventually the conclusions of this work as well as final considerations and future work are exposed in the *Chapter 14*

The *Annex* sections contain useful additional information to complete and to better understand this research work.

The *Glossary of terms* contains useful reference information of different terms related with this project work.

Part 1. New Hardware Architecture for Low-Cost Functional Test Systems

Chapter 2

Functional Test Basics

The target of this chapter is to briefly introduce the reader in the role of test inside the world of printed-circuit-board (PCB) manufacturing. The PCB manufacturing basics and processes are briefly explained in order to explain why test engineering has become so important to assure the product quality, it means that product works as expected when it leaves the factory, and product reliability, that refers to its resistance to failure in the field. Also the Test & Inspection basics are explained, as well as the technology trends that are making some old typical solutions not feasible to use in current production.

2.1 Basics of PCB manufacturing

The evolution in the electronics manufacturing industry has led in the past years to the huge emergence of the Surface-Mount Technology (SMT), that is a design standard for constructing electronic circuits where the components are mounted directly onto the surface of the printed circuit board (PCB). The surface mount devices (SMD) have small metal leads, also called pins that are soldered directly to the surface of the PCB on plated copper pads, called solder pads. The SMT technology has largely replaced the older Through-Hole technology where components with wire leads, thru-hole devices, are fitted directly into holes of the PCB. SMT provides a large catalog of advantages in front of older through-hole techniques:

- Smaller components lead to high integration and allow high density of components in the PCB.
- Simpler and faster large-scale automated assembly.
- Components can be fitted to both sides of the PCB
- Cost-related savings as SMD components are generally cheaper than through-hole counterparts.
- Possibility to integrate more functions onto ASICs and other complex devices with a large pin count, but small footprint space.
- Less parasitic (unwanted) effects when using high frequency signals through the PCB.

Anyway in some special cases, through-hole technology is still a necessity or offers an advantage in front of SMT, so it is still used in some modern electronic designs:

- The initial cost to set up the SMT manufacturing process is still higher, so in small production lots, it still offers an advantage.
- Some big components like connectors do not have a SMD counterpart, leading to the use of mixed-production processes where both SMT and through-hole mounting technologies are used.

- The emergence of the new Pin-in-Paste technique, to mount thru-hole components in SMT processes by also using solder paste to fill the drill holes, still have some important restrictions; because thru-hole components need to have high-temperature resistant housings, the components heights and pin length need to be adapted to the new process; the component must have a flat surface parallel to the mounting plane for suction nozzles and the packaging of components received from provider also needs to be adaptable to the pick&place unit, and sometimes the providers do not use tape-and-reel or tray packages for big components.

2.2 Basics of PCB Test & Inspection

The main difference between test and inspection is in the definition itself, as inspection only determines if the board *looks* correct but cannot verify if it really *works* correct as test makes. Only a true test process can establish if board functionality is the really expected, but inspections offer some advantages over test, as it does not need any "manufacturability" from the board under test, and essentially is non-invasive. As it can be seen in *Annex A: Mount process with SMT and HM for TV PCBs,* explaining the process flowchart of PCB manufacturing, test and inspections represent a trade-off, and must be combined to establish a successful test strategy.

As explaining in detail all different test&inspection strategies is not the main target of this project, so only a brief explanation will be made to settle the basic ideas.

PCB Automatic Inspection techniques:

- Automated Optical Inspection (AOI): In the simplest system, it consists of a camera and analysis software to make the pass/fail decision on some aspect of the board, from mechanical defects, solder paste placement, solder bridges, open and short circuits, missing components, component polarity, placement accuracy, etc.
- Automated X-ray Inspection (AXI): It is based on the same principle of AOI, but can be used to inspect balls under a BGA packages or not clearly visible components under shields.

PCB Automatic Test techniques:

- In-Circuit Test (ICT): It is a technique based on having access to the circuit nodes of a PCB, to check for shorts, opens, resistance, capacitance and other any analog parameters that can be considered to decide if board is ok or not. Typically the traditional ICT relies on bed-of-nails fixtures to check by probes all the accessible nodes of the PCB. Nowadays, flying probe testers, with quick probes capable to have access to every single place of the PCB, are gaining ground to the classic fixtures because of the greater flexibility offered.
- JTAG Boundary Scan: If an active component has JTAG capability, this architecture can be used to create stimulus in the board, and providing options to test interconnections between the boundary-scan component, typically the main microcontroller, and other active components like memories, logic cells, etc.
- Functional Test: It is a technique mainly based on emulating the final operation environment of the PCB in order to check if it has all the desired functionalities working as expected. It requires a deep knowledge of the product under test in order to simulate the final system, and check as much as possible. Typically some functional tests are also relying in bed-of-nails fixtures to access the nodes of the PCB, but as

new technology trends are making this access much more difficult, some other philosophies based on connecting the board exactly as it is made in the final assembled system are thriving.

As it is obvious, it is not common to rely only in one of these approaches to develop a successful test strategy, so although one of the non-written rules is to avoid looking for any problem more than once, usually these strategies are combined and overlapped.
It is accepted in test industry that finding a fault in a stage of the whole manufacturing process costs more than finding the same fault in the preceding stage. In general this is due to the extra complexity in the repair process that is added when product is nearer to its final state. For example in case of LCD TV, to find a defect at system-level implies the partial disassembly of the TV set, and either repair or swap the faulty board. If the fault survives all factory processes and arrives to the field, turning into a market claim, the costs can be still higher. In other cases, if the PCBs are produced for another OEM, it can lead to severe contract issues if defects go beyond the limits in the partnership agreement.

2.3 Mixed PCB Manufacturing processes for TV Main Boards

For the particular case of main boards production for LCD TV, that has been the main business of BCN Tec, there is still a big quantity of commercial TV input/output connectors of large dimensions, and that's the reason that mixed production has been used over the years, combining both SMT technology and through-hole mounting technologies.

Figure 2.1. BCN Tec AutoMount Production Area

Although in recent models this trend is in recession, mounting bigger-sized components in the SMT lines instead of in the HM process, *Annex A: Mount process with SMT and HM for TV PCBs*, introduces the usual mount process in two phases to mount the leading product in the plant, the TV main board PCBs.

2.4 Technology drivers for test evolution

As it can be seen in the process flowcharts shown in *Annex A: Mount process with SMT and HM for TV* PCBs, there is no place in BCN TEC for some typical test strategies like ICT, that have been in regression over the last years as new technology trends are affecting the way the products are tested. A brief summary of these technology drivers is exposed, and the BEN board, the main LCD TV board for Sony FY2010 low-cost LCD TV chassis, called with AZ1N codename, is taken as an example to see how these trends are present in nearly every high-end product.

Loss of visual access for AOI

Area array packaging technologies for multi-leaded ICs, like Ball Grid Array (BGA) are making life difficult for visual inspections, because there is no visual access to inspect solder joints. The RF shielding for high-speed signals and heat sinks for main microcontrollers are also starting to cover large areas of PCBs reducing the coverage that can be achieved by relying only in visual inspections.

In the case of BEN Board shown in Figure 2.2, the main microcontroller and the two DDR2 memories have a BGA package that makes impossible to inspect solder joints by AOI. There is also a problem with the HDMI connectors that are mounted in SMT process, because the connector body itself obstructs the point of view of the inspection cameras, which are placed in the vertical axis. In this board heat sink was finally not mounted although it was prepared in the board layout, as it was not needed after thermal tests in the main microcontroller. Anyway, it is not possible to rely on this to create a test strategy, as heat sink can be mounted at any time as a running change if some market problems because of temperature appear.

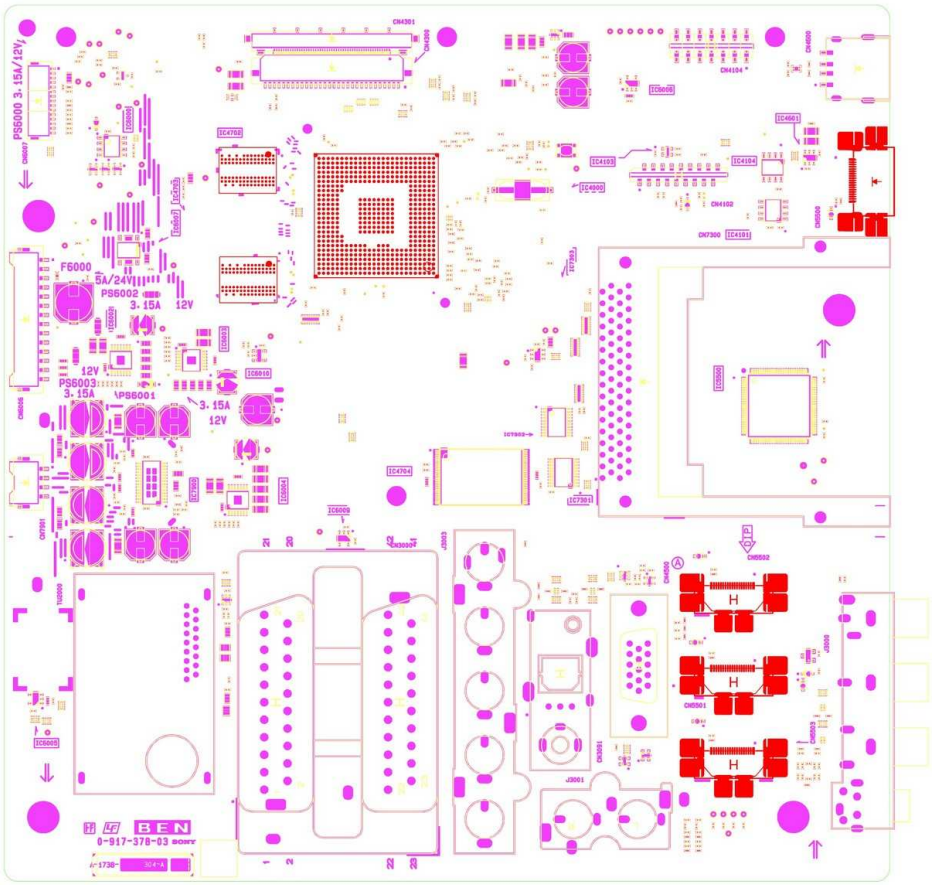


Figure 2.2. Top Side view of components (remarked in red) in BEN board with no visual access to solder joints with a vertical field of view (FOV)

Pin density driving loss of bed-of-nails access

The increasing density of interconnections is making physical access to the signal nodes by using bed-of-nails technology more constrained, as there is no real space for test pads or jiglands in the PCB. It is also not recommendable to rely on board designer to put jiglands on all signal nodes, as design for manufacturability is not always the main priority, and it is nearly impossible for designer because of lack of space in the final artwork.

As it can be seen in Figure 2.3, the BEN board only measures 193 mm x 191 mm, and there are too many signal buses coming from the main microcontroller on the bottom side. The number of jiglands of 1.5 mm is very reduced, as low as 32. That means a very low ICT coverage, and also limited possibilities for functional tests made by only using jiglands in bottom side.

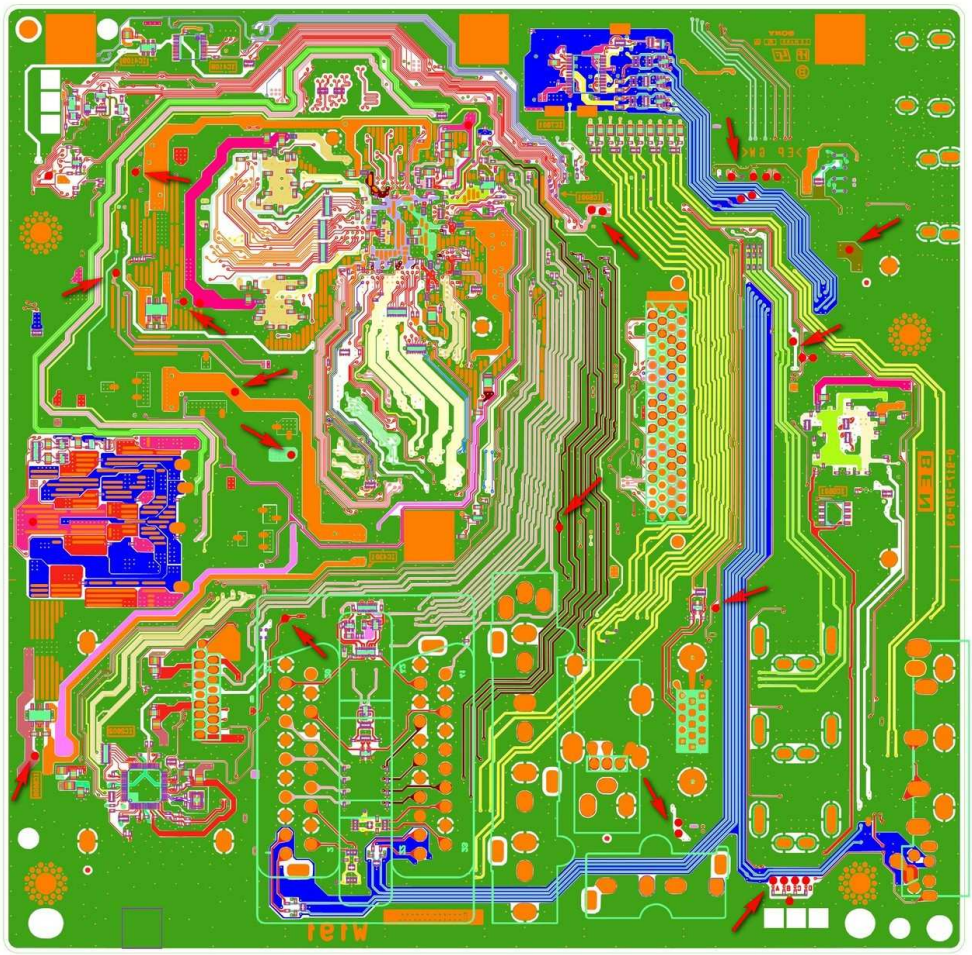


Figure 2.3. Jiglands in Bottom Side of BEN board (remarked in red) and signal density around main BGA

Loss of bed-of-nails access at High Frequency

Physical access constraints are only the first dimension of the access problem, as higher frequencies and differential pair signals demands controlled impedance, and the minimum discontinuities in the signal. The risk to worst signal integrity if jiglands are used in these signals usually means that design usually decides to not grant access in order to not take extra risks on the product long-term reliability, reducing again substantially the test coverage.

In the BEN board shown in Figure 2.4, it can be clearly seen as there is no jiglands in the high speed or differential signals, for example, differential signals from HDMI connectors to the internal HDMI switch, from HDMI switch to main microcontroller, or from main microcontroller to TCON connector for LCD panel. Of course there is also no jiglands in the signals to DDR memories that requires length matching and controlled impedance, and for signals from main microcontroller to NAND Flash.

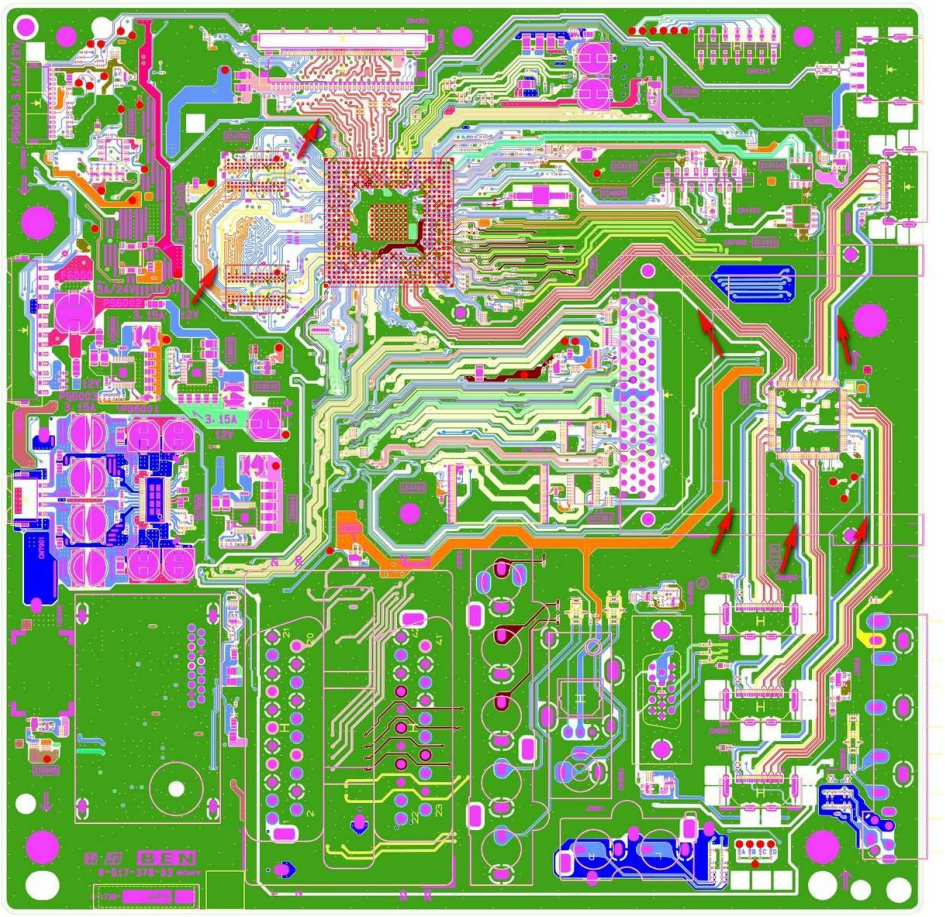


Figure 2.4. High frequency or differential signals in Top Side (remarked by red arrows) with special routing techniques like mitering to preserve a good signal integrity

Increasing board complexity

As the number of solder joints per board increases, so does the potential for faults, increasing the necessity to rely on mixed test and inspection techniques, not only to filter the problems, but also to improve the process as the vast majority of faults are structural and to have measurement data is vital to drive process improvement. There is also useful to note the challenges that SMT process generates to test processes, for example the trade-off in solder paste composition, because an increase in flux percentage that could be useful to decrease the number of shorts, can increase also the NFFs (No Failure Found) count in a bed-of-nails ICT fixture, as the flux creates a resin cover in the jiglands causing contact problems.

In the BEN board shown in Figure 2.5, the number of apertures to put solder paste in the top side stencil rises to 2941 pads where solder paste is applied, and the number of components in top side to 491. Of course that does not mean that all parts present in the layout are finally mounted, as there are some components of the PCB that are only present to have several mount options available as running changes, but anyway it gives an order of magnitude of the number of shots, and the increased complexity of the board. This data, gives a brief overview of the high complexity that would have a bed-of-nails ICT fixture, because of the huge number of nodes to analyze, and this is only in an imaginary case that all nodes would have an associated jigland, something that does not happen in the BEN board and in many other high-end products.

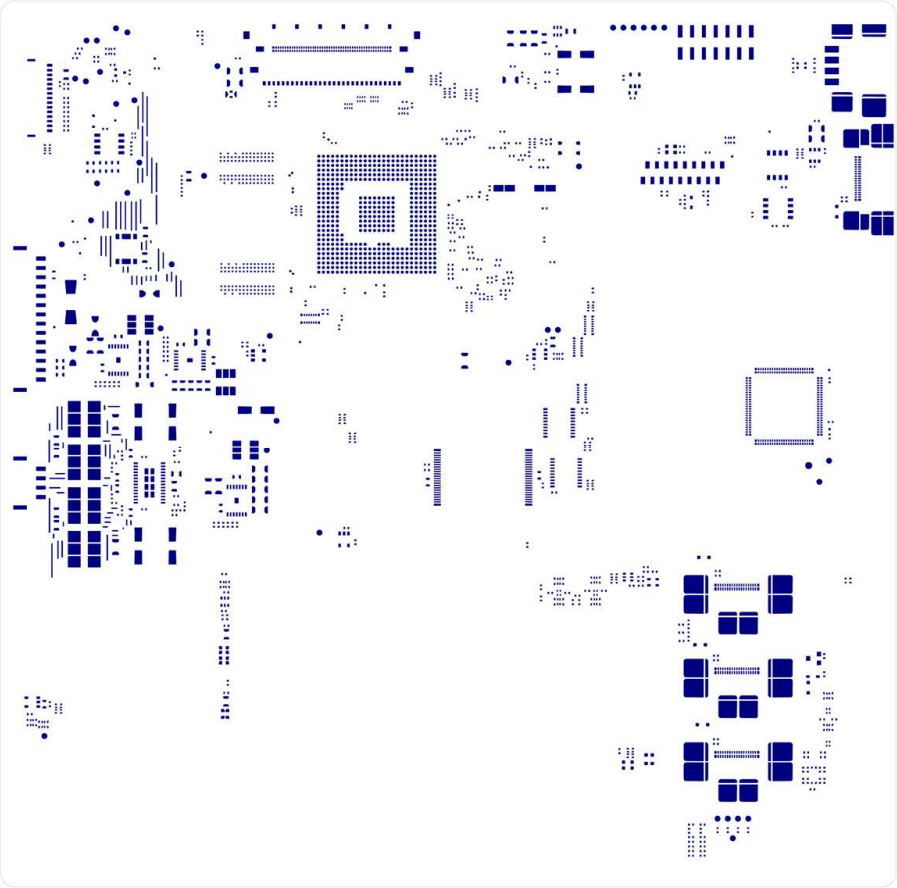


Figure 2.5. Apertures in Top Side solder paste layer

Security issues in high-end products
As the access to mass consumer electronics technology is becoming widespread, the security issues to protect the products against illegitimate use are also making life difficult to test engineers, as some typical test methods are restricted because of locked access to the low-level hardware.
In some cases it is only a hardware limitation because in Mass Production phases there are no connectors or jiglands to access the typical debug or service interfaces like JTAG, serial ports, etc. In more restricted cases the access to some interfaces like JTAG is physically locked in Mass Production version of the main ASICs.

In the BEN board, the JTAG access to main microcontroller is hardware-locked from the ASIC foundry, so test methods like boundary-scan are completely discarded.

Variability of Defect Spectrum
As the complexity arises around all manufacturing processes, it is not strange that the defect spectrum for any PCB would have lots of different problems and will not be concentrated only around one defect. These problems comes from the strictly electrical ones that come from a non-working component, to the process-related ones, like opens, shorts, bent pins, missing components, not enough solder, etc. The result is that no single technology provides full coverage and rely on mixed techniques is the best way to create a successful strategy.

The AZ1N BEN Board can be seen as an example of the necessity of mixed solutions test strategy. In the Figure 2.6 there is a pareto diagram showing the fault spectrum stats four

weeks after mass production start, in the first ramp-up of production, and can be clearly appreciated the extensive range of defects and components affected. From 33975 boards produced, 67 defects were found, giving a high rejection data of 1972 ppm. Also in these chassis the defects of IC4000, the main microcontroller that at first were assigned to process faults, finally resulted in a problem of the part itself, raising the number of electrical problems only detectable with test instead of optical inspections. In the Figure 2.7 there are some typical defects produced in SMT process.

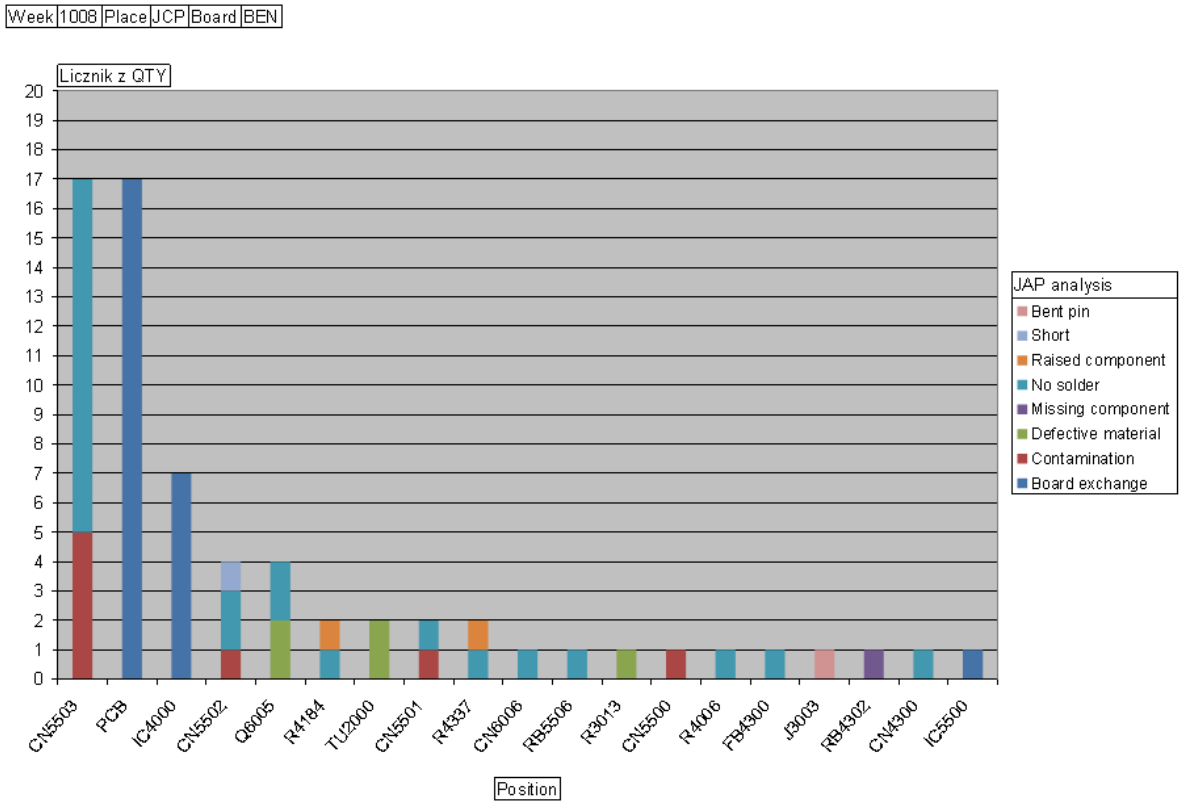
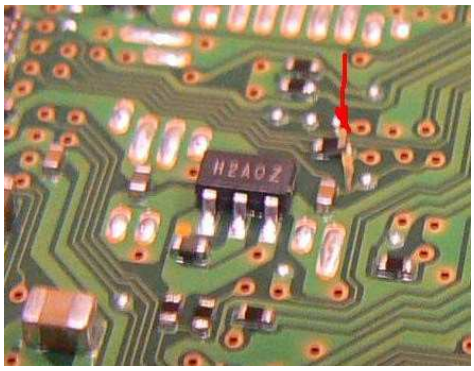
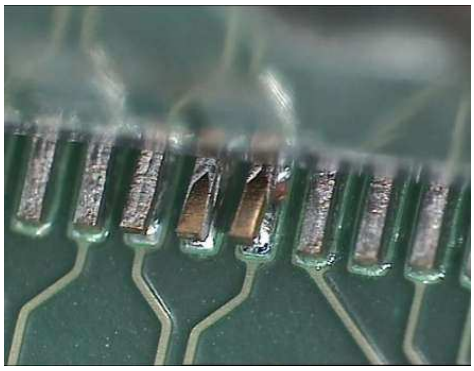


Figure 2.6. Pareto diagram showing weekly reparation data in Jabil Circuit Poland BEN board production

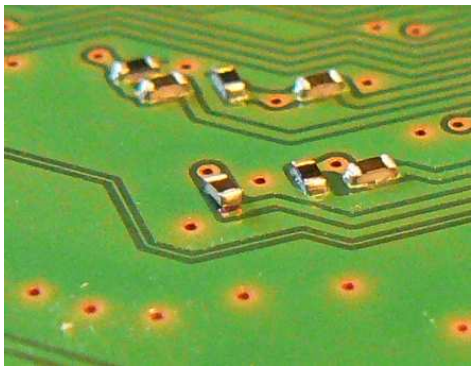


(a) Contamination

(b) Shorts



(c) No solder



(d) Raised component

Figure 2.7. Examples of typical defects in SMT process

2.5 Functional Test at board-level as the best solution to test necessities

The technology trends that have been analyzed are becoming widespread in all the state-of-the-art products in the electronics industry, and functional test can be clearly seen at this moments as the best solution to offer quality test services, with the possibility to offer a board coverage that it is worth the amount of effort needed to develop these systems. It has been seen that bed-of-nails fixtures that in the past were dominating the test market, are also in regression because of the high complexity and high NFF count problems. Automatic functional test solutions like the used in BCN Tec facility, based on connecting the external board connectors, making a simulation of final condition in the system-level assembly, are possibly the best solution to offer a balance of respectable board test coverage and NFF ratio.

Of course the widespread implementation of functional test for any kind of boards has a big impact in budgets because of the typically high cost of test equipment necessary, and also the high cost of automation needed in the process. It is also very important to mention the high knowledge of the product under test that is needed to apply successfully these functional test strategies. In the next chapter these two issues will be deeply addressed, by focusing on the equipment and the knowledge needed to develop a functional test strategy for main boards of LCD TV.

Chapter 3

New Hardware Architecture for Functional Test of LCD TV main boards

As the current high-volume product mounted in BCN Tec factory is still the main board of an LCD TV, the work will be focused on this product. Also because it is the most technology challenging product and the product with a higher know-how in the factory about it, it will be selected to demonstrate the feasibility of the development of new low-cost customized test equipment.

In this chapter, the initial work will be focused on analyze which ones are the test features needed in that selected product, main boards of LCD TVs. After this analysis the new hardware architecture based on the development of customizable embedded systems with a platform approach is presented, and a brief economic assessment is made to compare the advantages of the proposed customized hardware architecture with the current one.

3.1 Introduction of LCD TV main boards testable features

An LCD TV is a complex product that is just now reaching his maturity, and of course it has evolved a lot in the last years from his introduction on the early years of this decade, to his takeover as the leader in the television market in 2006 in front of CRT and Plasma displays. First models functionality was exploded in several boards, each one taking care of different functionalities, for example analog and digital blocks, audio processing blocks, etc. The trend of course was to first integrate as much as possible in the main board, and later to introduce as many processing cores as possible in the main microprocessor. That left the current situation, where so many functions are dealt by an advanced System-on-Chip (SoC) specifically designed for LCD TV architecture. The main controller deals with digital audio/video decoding, analog audio/video decoding, video and audio processing, HDMI and LCD display connectivity, etc., and also has advanced microcontroller features, like interfaces with DDR NAND flash, or any other peripheral devices. A block diagram of a SOC with a large number of functionalities used in TV boards is shown in Figure 3.1.

In fact there are still some external peripheral devices, but this is the trend that is allowing more and more integration in every new chassis, and drives all the new features that are typically appearing in TV market. That high integration of features and the big amount of input and output interfaces present in the main board of a LCD TV, makes the test of this product a complex process, as the list of functionalities is very extensive.

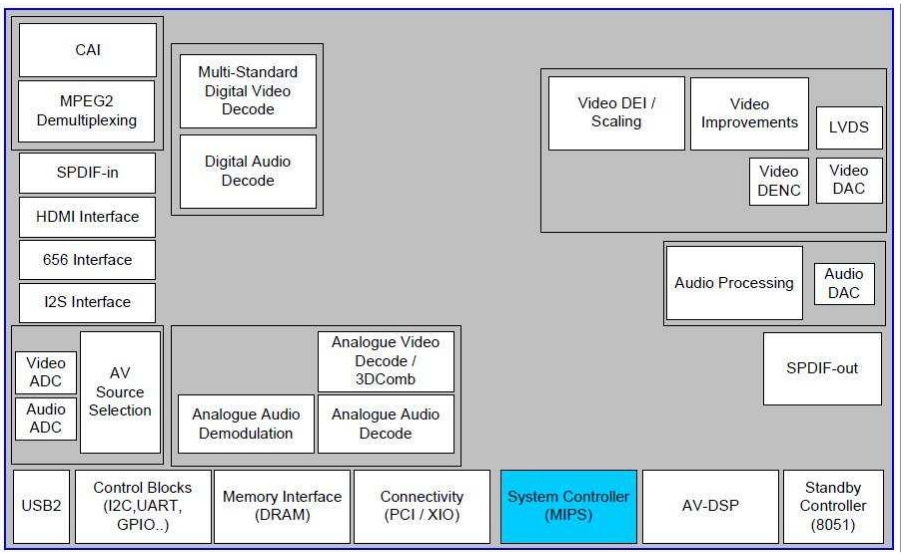


Figure 3.1. Block diagram of PNX8542 TV SOC

In Figure 3.2 the block diagram of a main board for LCD TV is shown, specifically BAL board for Sony AZ1L FY2010 high-end chassis, which is taken as example because that high-end chassis have all modern state-of-the-art functionalities currently found in LCD TV market.

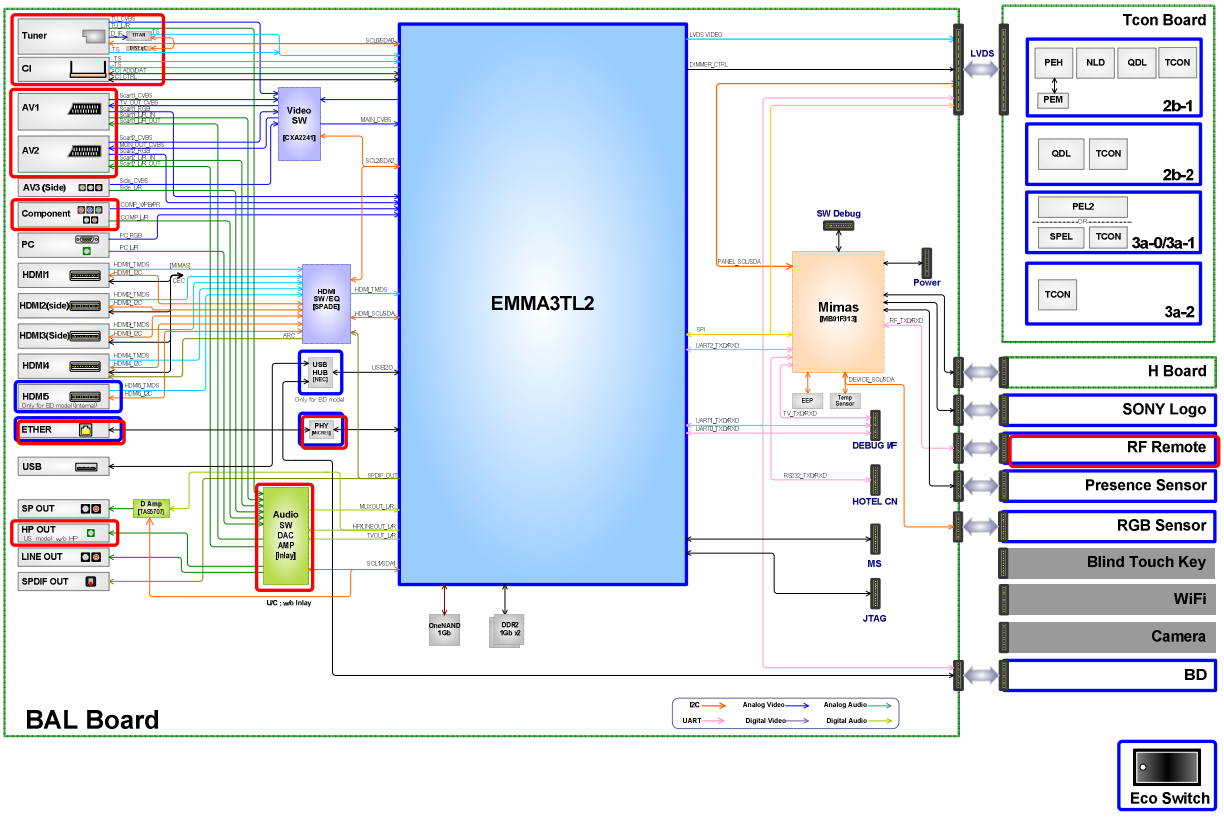


Figure 3.2. BAL Board Block Diagram

From these block diagram, a tree view structure of all test capabilities needed to make a full functional test has been summarized:

- Signal Generation (Inputs to Main Board)
  - Analog Video Generation
    - VGA

        - RGB
        - YUV
        - CVBS/YC
    - Analog Audio L/R
    - HDMI 1.4
    - SIRCS Emitter (Infrared)
    - CEC Protocol Generation
    - I^2C Master Generation
    - Programmable Outputs (PIO)
    - Serial Port UART
- Signal Acquisition (Outputs from Main Board)
    - Analog Audio Analysis
    - SPDIF Digital Audio Analysis
    - LVDS DesSerialization & Capture
    - Analog Video Capture
    - Voltage/Current Measurements
    - Programmable Inputs (PIO)
- Power Supply
- RF Signal

All these requirements of functional test make impossible to find in any individual commercial equipment, and that means an important investment in mixed test equipment to make an automatic test machine with the possibility to achieve maximum coverage when making functional test.

3.2 New Hardware Architecture Proposal

As current automatic test systems, known as Circuit Board Adjustment (CBA) machines, used in BCN Tec are able to achieve nearly maximum test coverage, the comparison of the new hardware architecture will be made with them.
In Figure 3.3 there is the block diagram of the hardware architecture of the current CBA. As it can be seen the architecture consists on three levels:

- Equipment. To generate or analyze all the signals or stimulus needed for testing.
- Mux Interface. To switch audio/video signals from equipment to the device under-test (DUT), or the inverse path from DUT to analysis equipment.
- Automatic connection unit, that is the only part customized for every different kind of board.

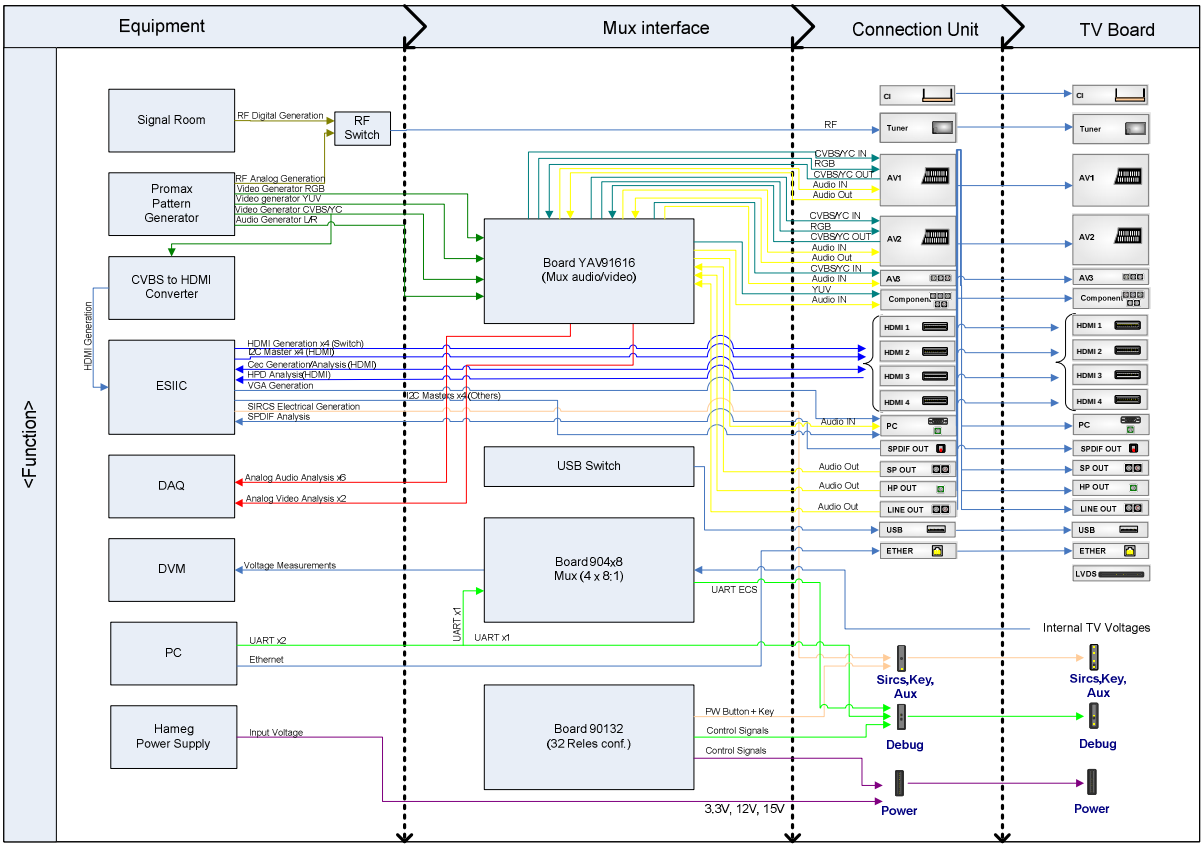


Figure 3.3. Current CBA hardware architecture

In the new proposal, the cornerstone is the reduction of common commercial equipment from the old CBA. The typical idea, not only in the current CBA but also in all the functional testers in the market, is to have commercial off-the-shelf equipment ready to give solutions to any test necessity, and this equipment have to be multiplexed with a customized hardware interface to reach the board under test. The philosophy of the proposed new hardware architecture is to put the functionality as near as possible to the board under test, customizing to real necessities not only the interface but also the equipment itself.

The basic idea is to have two smart embedded systems with all the functionalities really used of the old commercial equipment and an ability to reconfigure them and reuse in several other projects. A third board will be used to manage the signal path from the device under test to the two smart boards.

In Figure 3.4 there is the block diagram of the proposed hardware architecture for the new CBA low-cost.

As it can be seen the architecture consists on the same three levels but this approach gives some added advantages:

- Common equipment reduced to only one PC for control issues and one generic Power Supply in comparison with all the hardware in current CBA.
- Peripherals reduction in controller PC, like digital multimeter (DVM) and data acquisition cards (DAQ).
- Possibility to use only one PC as a controller of multiple test positions with independent hardware.
- Smart equipment inside the base unit instead of the non-smart equipment of the base unit, only used for multiplexation or interface issues, in the old CBA

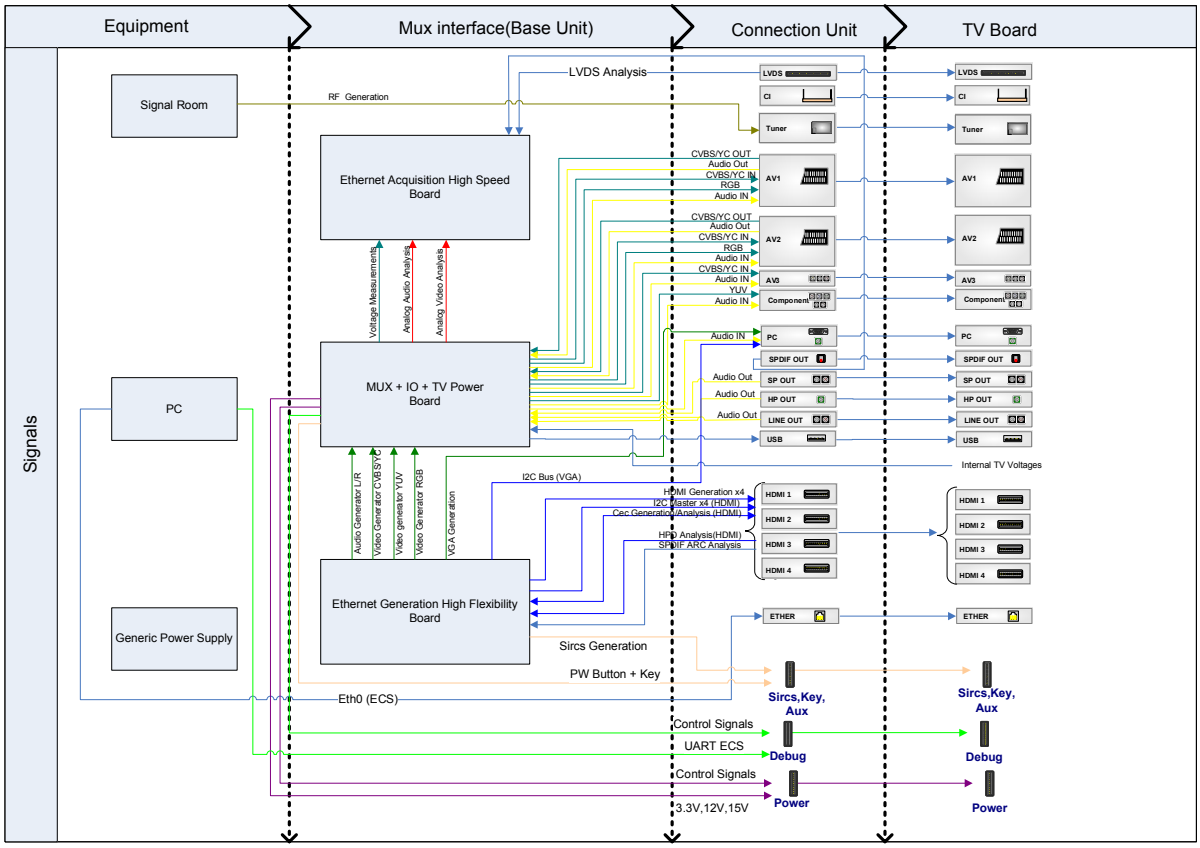


Figure 3.4. New CBA Low-Cost hardware architecture.

3.3 New Hardware architecture economic assessment

Although this project is structured from a technical point of view, some numbers are presented to be able to evaluate the benefits of the new hardware architecture for the low-cost functional testers. Again it is important to remark that this new hardware architecture is only a part of a global strategy to reduce the functional test systems cost, that also includes a redesign of mechanical and software architecture, although the biggest cost savings are achieved with the hardware architecture redesign.

The initial calculations have been made making some assumptions:

- Fixed costs for prototypes development, board series manufacturing, etc., (NRE,Non-Return Engineering) have not been considered.
- Cost of the boards has been roughly calculated, making assumptions on factors like blank boards cost, embedded system architecture selected, final key components selected or manufacturing costs.

3.3.1 Cost estimation for common parts of automatic test machine. Common rack block

The common rack or pressure unit block are all the parts, hardware and mechanical ones, that are classified as common equipment of the automatic test machine, it means parts that are not changed depending on the DUT. The common rack block has a common interface with the test fixture block that certainly needs to be changed depending on the DUT.
In Table 3.1 and Table 3.2 it can be seen how the big reduction in commercial equipment affects in the final price of CBA rack. In fact in CBA Low-Cost there is no common

instrumentation inside the common rack except for the host PC. Although is not the main subject of this project, more focused on hardware architecture issues than software issues, it is useful to explain that the host PC acts only as a main controller, so it does not have any hardware like data acquisition boards or digital multimeters inside to be shared between several test slots, allowing the parallel control of several positions with the same controller, and reducing not only the number of computers needed but also the single position software licenses for proprietary Test Management software or image processing proprietary software to decode the different identification labels in the PCBs.

PRESSURE UNIT EQUIPMENT DETAIL

Equipment for 1 CBA

Concept	Cost
Computer equipment & software	
Digital Multimeter PCI card	1.196 €
UPS	140 €
License Matrox Imagine Library 9.0	338 €
PC+Monitor+Keyboard+Mouse	850 €
RS232 expansion card	200 €
Ethernet switch	13 €
PCI card DAQ	418 €
PCI card CAN bus (MM7735702)	1.069 €
CBA generic boards & cards	
Switching multiplexer board (Sistel YAV904X8)	888 €
Switching relays board (Sistel YAV90132)	861 €
Communication interface board	250 €
PIO 8 out (Sistel MW0604PZ)	91 €
PIO 8 in/out (Sistel MW0304)	61 €
Audio / Video switching matrix board (Sistel YAV 91616)	1.017 €
ESIIC board (Sony A/E)	751 €
ALTERA DK-NIOS-2C35N board	680 €
Standard equipments	
Programmable power supply HM7044	1.250 €
TV Pattern Generator PROMAX GV798	3.210 €
SCART to HDMI converter	110 €
Other	
Rack Wiring (CX-NEVITNEWVIT.C01)	551 €
RF Switch board + DAQ Adapter board	132 €
Rack Assembly subwiring (25h x 19€)	475 €
Total	**14.551 €**

Table 3.1. BCN Tec current CBA Pressure Unit Cost

RACK EQUIPMENT DETAIL

Common Equipment for 5 slots new CBA

Concept	Cost
Computer equipment & software	
UPS	140 €
License Matrox Imagine Library 9.0	338 €
PC+Monitor+Keyboard+Mouse	850 €
Ethernet switch	13 €
Standard equipments	
Power supply 24V 20A QS20-241 RS:6700677	278 €
Other	
Rack Wiring (CX-NEVITNEWVIT.C01)	551 €
Rack Assembly subwiring (50h x 19€)	950 €
Total	**3.120 €**

Table 3.2. Proposed CBA Low-Cost Rack Unit Cost

3.3.2 Cost estimation for non-common parts of automatic test machine. Test fixture block

The base/slot unit equipment are all the parts, hardware and mechanical ones, that are not classified as common equipment of the automatic test machine. That means all the parts depending on the DUT that need be changed for every different board introduced in production. The philosophy behind this approach is that common equipment is reusable for every different board, and base unit / slot fixture needs to be redesigned for every new model, although there is always a common interface between the base unit / slot fixture and the common rack.

In fact, for the LCD TV product the main parts of interface boards are reused, and it only needs a redesign if the new model introduces new features. The major changes are typically mechanical ones like new positioning of the connectors in the TV board, and that reflects on the cost, anyway there is not a big difference from the old approach to the new one as it can be seen in Table 3.3 and Table 3.4.
However the main change is in the boards, as the cost of interface boards in the old CBA plus wiring is really very similar to the cost of new “smart” boards that really not only includes the interface from common equipment to the main connection unit, but also include the functionality itself, so in the sum of the common block cost and test fixture block cost, the cost savings will be easily appreciated.

BASE UNIT EQUIPMENT DETAIL

Equipment

Concept	Cost
Base Unit Mecha. Parts & Wiring	
Mechanical Base	5.744 €
Wire set for Base	543 €
Wire Assembly	269 €
Standard equipment & connection boards	
CAM PCMCIA	43 €
USB Camera	47 €
Illumination Board	8 €
Usb Board + Connector	6 €
USB Protection Check Board	64 €
Interface boards	
Intermediate board for vitam (SIRCS, POWER, SPK) + Connectors	35 €
DAQ Boards	50 €
HDMI to VP-F Interface Board	201 €
Panel Matrix Interface Board	725 €
Panel Esiic Interface Board	451 €
Panel Mux Interface Board	222 €
Total	**8.407 €**

Table 3.3. BCN Tec CBA Base Unit Cost

SLOT EQUIPMENT DETAIL

Slot Equipment

Concept	Cost
Slot Unit Mecha. Parts & Wiring	
Mechanical Base	4.885 €
Wire set for Base	543 €
Wire Assembly	269 €
CAM PCMCIA	43 €
Standard equipment & connection boards	
USB Camera	47 €
Illumination Board	8 €
Usb Board + Connector	6 €
Intermediate board for vitam (SIRCS, POWER, SPK) + Connectors	35 €
USB Protection Check Board	64 €
New instrumentation boards	

Ethernet Acquisition High Speed board	350 €
Mux Video/Audio/DVM, Relay	456 €
Mux + IO board	540 €
Power switching board	279 €
Total:	**7.524 €**

Table 3.4. CBA Low Cost Slot Unit Cost

3.3.3 Cost comparison summary

In the Table 3.5 and Table 3.6 there is the summary of savings that can be made by using the new CBA architecture proposal instead of the current one. It is important to remark again that this summary have been made only focusing on hardware proposal of improvements, without introducing the mechanical and software improvements that will also lead to more savings. Anyway it is enough to initially demonstrate the economic feasibility of the project.

Only 1 Slot/Machine	Current CBA	CBA Low-Cost
Common Equipment	14.551 €	3.120 €
Slot Equipment	8.407 €	7.524 €
Total:	**22.958 €**	**10.644 €**

Table 3.5. Cost comparison between for only one unit.

5 Slots/Machines	Current CBA	CBA Low-Cost
Common Equipment	72.755,00 €	3.120 €
Slot Equipment	42.035,00 €	37.620,00 €
Total:	**114.790 €**	**40.740 €**

Table 3.6. Cost comparison for 5 slots CBA Low-Cost against 5 current CBA machines.

Chapter 4

Embedded system architecture selection

In LCD-TV Business market the product lifetime is very short, usually products not lasting in mass production more than one year, that means that typical production life cycle is to introduce at least a new TV model every year.
Also in Electronic Manufacturing Services (EMS) market were customers are asking the providers to adapt their processes to new products in a whisker, the ability to manage sudden manufacturing process changes and to be prepared to adapt to new products in a short time, is key to attract new customers.

In this environment, where usually quick changes in capabilities of the equipment for functional test are needed in a short time, as every new LCD-TV chassis or any other product introduces new state-of-the-art hardware and software features, and with the big constraint of cost restriction in new investments, the most important strategical decision is about selecting an optimal architecture for the homemade smart test equipment that will be used in the functional test machines and is the cornerstone of the new hardware architecture.

As it has been already explained, this new CBA low-cost hardware architecture is part of a bigger project focused not only in a new hardware architecture but also on mechanical and software improvements focused on reducing the costs. From now on, this chapter will only focus on the definition of the smart embedded systems needed to take charge of all the functionalities offered by the commercial test equipment.

The selected technology must be appropriate to design an open, flexible and configurable hardware platform reusable for any kind of equipment for functional test. This “platform” design approach is one of the key points to be considered, as it has to be scalable and upgradeable to new needed features or functionalities in a short time, and also the effort to run this project should be reused in future designs with same design philosophy.

4.1 Embedded system required features. Decision metrics evaluation

As the strategical decision about system design has a big impact in project cost and time scale, a lot of factors has been deeply analyzed.

First of all, some metrics to compare different solutions are defined and finally, it is decided which one adapts better to the required environment. It will be made by examining which of these metrics are the most important to our project. As typical in this kind of decisions some trade-offs must be made.

- **Non-recurring Engineering (NRE) Costs.** NRE costs refers to all one-time costs when designing system. In this project NRE costs go from worker hours during project development time to all equipment necessary for development, including software tools and new hardware equipment necessary to test the system. All the one time costs of fabrication of the product also should be included.
- **Lost opportunity costs.** These costs are associated with long development times or inflexibility that can make the project fail if the system is not able to adapt to a

production necessity at the right time. Sometimes in the desired platform design it means that current platform is not able to adapt to a needed new test feature, so a system architecture for the mid-term and long-term future will be required.

- **Unit costs.** The unit cost of the selected main device is the typical that most people think first, but the general cost of all the system including all discrete component must be evaluated through an estimation of the Bill of Materials (BOM). In typical low-volume applications like test equipment the unit costs are not as important as for products directed to end-user. Furthermore, the Minimum Order Quantity (MOQ) of the main system components is another factor that cannot be diminished, as expensive components disadvantage can be reduced if MOQs are lower. Also as more generic is the system architecture, more main components can be reused for multiple applications, and the risk of scrap components with no possible reuse is lower.
- **Production costs.** Manufacturing costs involves from prototype production to final product production and is important to identify if technologies used in the system architecture design will affect negatively to future manufacturing costs.
- **Design Reuse & Portability & Intellectual Property (IP).** The common practice is to use a device and in this case also a system architecture in as many projects as possible. The more generic is the system, the more projects it can be used on. Typical design reuse is achieved with code portability in systems based on generic microprocessors (generic uC) and Intellectual Property (IP) cores in systems based on programmable logic devices (PLD).
- **Time-to-market (TTM).** Time-to-market is the length of time from the system being conceived until its being available to use. A correct time-to-market will minimize NRE costs and lost opportunity costs, as project delays can be costly.
- **Time-to-prototype.** Time-to-prototype is a subset of Time-to-market but it is important to consider it alone as the prototyping phase is one of the major project milestones, because with the current investment downtrend the prototype impact is one of the major factors to project approval and continuity in the future.
- **Flexibility.** The flexibility of the selected system architecture must be enough to adapt to the already presented scenario of constant changes in tested products and production test process.
- **Reliability.** The reliability is the probability that the system will not fail.
- **Maintainability.** The maintainability is the ability to resolve some issues with the embedded system in a short time-frame, and depends directly on system complexity.
- **Performance.** The performance will be measured not only for the current capabilities that the system must achieve but also for future upgraded features.
- **Size.** The size of main system components used in PCB board can be considered in cases of space restrictions.
- **Power.** Power consumption.

The metrics that have been defined should be weighed in order to quantify the impact that each one will have in the final decision. The environment of the LCD-TV production and new EMS challenges had already been described, and now each one of that real necessities of the test equipment will be weighed with the previously defined metrics, in order to take a correct system architecture decision in the early stages of design.

Basically the final test equipment quantities will be very low in comparison with quantities of a typical end-consumer product, and this fact is minimizing a lot the impact of unit costs and production costs. In the opposite direction the NRE costs will have a big impact in the total costs and it has been reflected on the final weights.

Again, in the already explained production environment flexibility and design reuse has an increased importance in front of nearly worthless factor like size integration or power consumption issues. Also in a production environment, by nature the reliability always has to be top class and above maintainability, basically because the production savings are always the priority, and the needed maintenance effort will be made without too many consideration if it is really needed to minimize the production stoppages.

Finally the performance has not been specially considered because initial short-term objectives are already clear, and all the considered technologies should reach it without too much setbacks. Anyway it is important that selected technology has a margin to still reach higher objectives because that will open further developments in the same platform..

Metrics	Weight	Comments
NRE costs	15	In current investments downtrend, the NRE costs have a big impact on decision.
Lost-opportunity costs	5	Long-term future adaptability above short-term results.
Unit costs	1,25	Low-volume quantities minimize that factor.
Production costs	1,25	Low-volume quantities minimize that factor.
Design Reuse	15	Design re-use is important not only for future projects but also to take advantage of current technical solutions in this project.
Time-to-market	15	The impact of this project should be reached as shorter as possible to better prepare for uncertain future.
Time-to-prototype	5	Minimized because in a “platform” design it is not going to be major changes from proto to final product.
Flexibility	15	Highly important factor in line with already explained production scenario.
Reliability	10	More important in a production environment, than for typical end-consumer products.
Maintainability	5	Very dependant of complexity, but in line with the flexibility.
Performance	10	Performance issues are important but all platforms are prepared to reach the objectives, and surpass it is not a priority.
Size	1,25	Minimized impact when comparing any solution against commercial equipment. Any embedded solution under study is going to be smaller in size than a commercial test equipment, so although the small space is important, it does not have an impact on the decision between embedded systems.
Power	1,25	Minimized impact in a production environment

Metrics	Weight	Comments
		compared with typical end-consumer oriented products with more restrictions.

Table 4.1. Decision metrics weights.

4.2 Introduction to embedded system technologies

After defining the metrics, the typical technology choices in embedded systems designs are summarized, in order to establish which ones are going to be the considered as options for the design system architecture.

- **Application-specific Integrated Circuit (ASIC).** An ASIC is an Integrated Circuit (IC) specifically customized for a particular use. The ASIC is defined as a product designed and used by one customer.
- **Custom Processor/DSP also known as Application-specific Standard Product (ASSP).** A Custom Processor is like the ASIC, an IC specifically designed for a particular use or application. Anyway, unlike ASICs this product is later generalized to use by many end customers and is made available as off-the-shelf components.
- **Programmable Logic Device (PLD) / FPGAs.** A Programmable Logic Devices are completely opposite to the ASICs because by their very nature are not specific devices targeting one application. The defined function can be given according final product necessities and is completely reconfigurable over the time. Field-Programmable Gate Array (FPGA) is the most commonly used subset of PLD family, and will be used without distinction instead of PLD in this document.
- **Generic Processor.** A generic processor that is not targeting a particular use or application, main difference compared with a custom processor. Typically a general purpose microprocessor available off-the-shelf with multiple uses and customers, and ready to use in as much applications as possible.

From all the exposed technologies the ASICs have been soon discarded without necessity to make a feasible proposal of system architecture, because NRE costs will be extremely costly and impossible to recover with so low volumes. The long time-to-market also made the use of an ASIC as main device of our system design completely unfeasible.

All the other available technological options will be combined in order to make at least, a couple of proposals for the embedded platform and these combinations are going to be evaluated according the defined metrics.

4.3 Embedded system architecture proposals. Combining technologies

4.3.1 FPGA+Soft-core+Embedded Peripherals+External Peripherals

Programmable logic solutions as FPGAs provide by nature a flexible architecture with the ability to add features over the time and at a low cost. An emerging trend in this market in order to further increase that flexibility, is to implement complete microprocessor architectures, usually called soft-core microprocessors, inside the FPGA chip.

The soft-core uC approach gives the possibility to completely customize the microprocessor to meet the exact design requirements, with no wasted or unused peripherals. Designing with this kind of system using a soft-core integrated into an FPGA will give the flexibility to implement functionalities in discrete system components, some in software some in FPGA-based hardware. Although it will increase the system complexity, with the system design flow including both hardware and software development, at the same time gives a perfect platform to avoid the obsolescence fears –the potential lack of future support for specific microprocessors, external controllers, memory devices, for example- by providing the ability to migrate designs to another devices without extensive modifications, preserving years of legacy code and development than can be reused because are not oriented to a single one external device.

Along with the soft-core uC comes a big number of system peripherals ready to use but some other user-defined peripherals can also be added easily to the system, increasing the legacy support to other IP cores already designed. The typical system components that will be first considered to create the basic core platform includes, memory controllers to interface with off-chip memories like Flash, SSRAM or DDR SDRAM and also communication interfaces like Ethernet media access controller (MAC) that provide a seamless interface to commercial Ethernet PHY devices. That Ethernet communication link should be enough to be used as control interface of the equipment, besides providing a high speed interface with host computer, enabling the possibility to develop high-end features requiring extra bandwidth, like video streaming or data acquisition.

The ability to create a general-purpose platform, that provides high performance with external off-chip memory and a reliable communication interface for every future application, is the main advantage of this proposal. This platform can be reused in several projects and reconfigured to match the specific requirements of each one, and this kind of adaptability is a huge benefit. The familiarity with design tools and high reuse rate will drive low the NRE costs over the long term, as well as minimizing the prototyping times and time-to-market times.

Some of the individual features that can be needed in the homemade equipment may need a dedicated hardware like ADCs, DACs, Video encoders/decoders, HDMI Rx/Tx, etc. In that cases the high-pin count of the FPGA and the unlimited flexibility to develop a controller for the device, makes this one the appropriate platform to increase the equipment functionalities easily, no matter which is the control interface of the external peripheral device. That architecture also enables the possibility to add new features to the platform by using a daughter cards approach, which means concentrating all unused pins of the FPGA in external connectors in order to connect extension cards that in the future will add new functionalities to the platform.

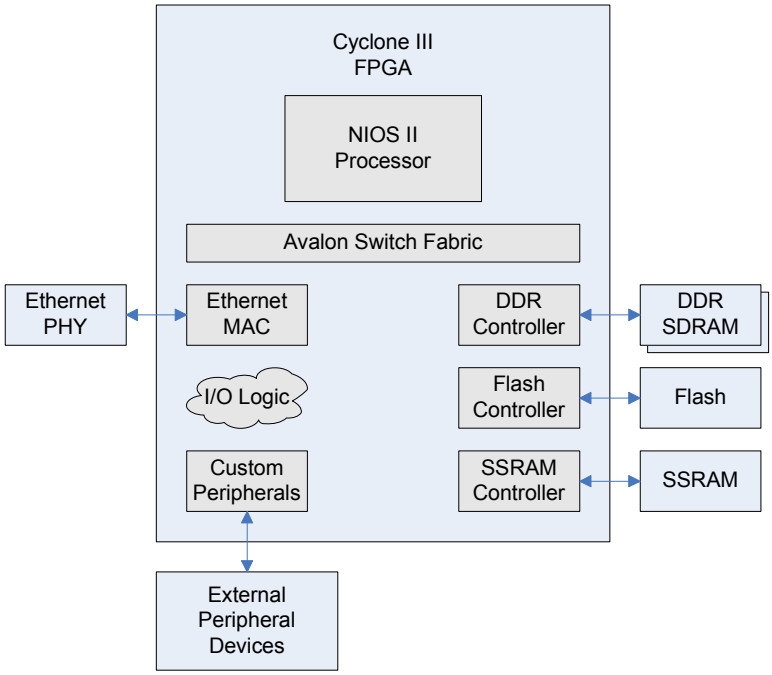


Figure 4.1. Block diagram for high-performance FPGA + soft-core architecture. Example for the Altera vendor case.

4.3.2 Generic Processor+Slave FPGA+External Peripherals

Generic or custom processor solutions provide by nature a very low cost compared with solutions based on large FPGAs. In case of a generic processor, the wide availability, and large collection of libraries and application notes, also give a big advantage in terms of development times, and that makes feasible to use it as a heart of an embedded system architecture for the test equipment.

However a solution based in a generic processor by nature only targets one single application or use, raising some concerns over the long-term future of an embedded system based only on this generic processor. To further increase the flexibility and ability to reuse the platform over the time, it has been considered to add an slave FPGA to the system, giving the possibility to implement some functionalities in FPGA-based hardware, that can be easily modified or reconfigured in case new necessities appear in the future.

The generic processor will come with integrated peripherals ready to use, although the ability to further add new peripherals on-the-fly that comes with the soft-core approach will be lost. The selected generic processor must have some number of essential peripherals needed to create the basic core platform, including peripherals for I2C and SPI communications, and Ethernet interfacing support to provide a link with the host computer for device control. That Ethernet communication link should be enough to be used as control interface of the equipment, but some doubts appear about the possibility to use this interface as a high-speed link, to enable high-end features requiring extra bandwidth, like video streaming or data.

The addition of the slave FPGA to this architecture based in a generic processor has been made to create a general-purpose platform, which again should be prepared to be reused in several projects. As explained before, some of the individual features that can be needed in the homemade equipment may need a dedicated hardware, and although the slave FPGA can be reconfigured to match some specific requirements, the generic processor will always

remain the same, and to upgrade it with new required interfaces or peripherals, a complete platform redesign will be needed. That is the reason of the inclusion of the slave FPGA, because it still opens the possibility to upgrade the features on the equipment by adding extension cards connected to the FPGA, as it has been explained for the previous architecture.

Although this architecture keeps unit costs very low in comparison with the other mixed alternative, there is some uncertainty over the long-term future of the platform and the general performance with high-end features.

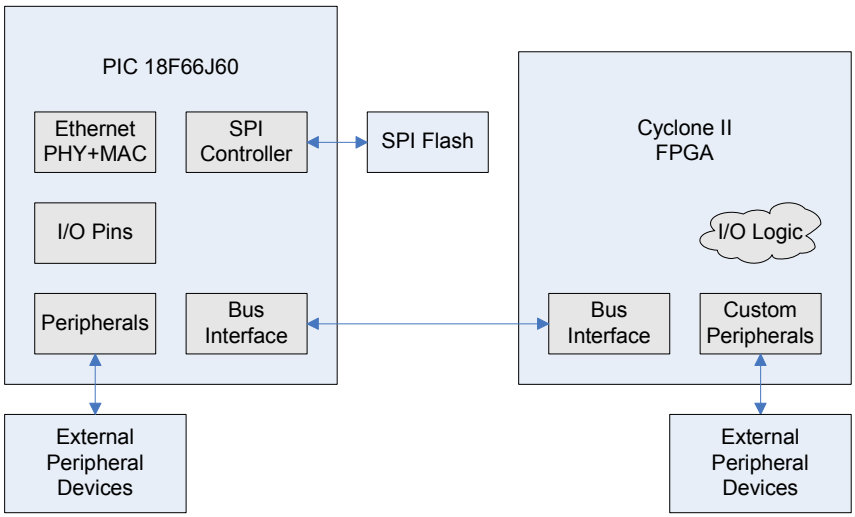


Figure 4.2. Block diagram for generic processor + low-cost FPGA architecture.

4.4 Embedded system architecture final decision

After defining the metrics and presenting different available options, a requirements priorization tool like Quality Function Deployment (QFD) will be used, in his simplest form, to help the decision-making process and to have a clear picture about which ones are the strongest points of every technology in order to be selected for the final system architecture.

	First level	ASIC	ASSP	FPGA	Generic uC		1. FPGA + Soft-core	2. Generic uC + FPGA
First level	**Weight**							
NRE costs	15	1	3	3	9		3	3
Lost-opportunity costs	5	1	3	3	3		9	9
Unit costs	1,25	9	3	1	9		1	3
Production costs	1,25	9	3	1	9		1	3
Design Reuse	15	1	3	9	3		9	9
Time-to-market	15	1	3	9	9		9	9
Time-to-prototype	5	1	3	3	3		3	3

Flexibility	15	1	1	3	1		9	3
Reliability	10	3	3	3	3		3	3
Maintainability	5	1	1	3	3		3	3
Performance	10	9	9	9	3		9	9
Size	1,25	9	9	1	3		1	1
Power	1,25	9	9	1	3		1	1
	100	**240**	**335**	**530**	**465**		**650**	**565**

Table 4.2. QFD chart for embedded system architecture selection.

From the QFD chart it is easy to see in which points a technology has a significant advantage over the others. From a first glance, it can be seen than the second mixed alternative based in generic uC, loses some of his advantages in NRE costs, as the one-time costs in development are increased, because of the added complexity to manage a bus interface between the main device and the slave FPGA. Meanwhile the added soft-core approach only helps to increase the flexibility of the FPGA-based alternative without losing any of their strongest points.

As it has been already commented the second option, generic uC + FPGA, has a point over the first option, FPGA + soft-core alternative, in unit and production costs, as the selected main components, of a generic uC and a small FPGA, are by far cheaper and easiest to mount than a large-scale FPGA device. Anyway as it has been reasoned before during the metrics weighing, the low-volume nature of this project makes some of this points less important than others, and the highest flexibility of the first alternative makes the decision easier.

It is good to remember that all this study and the final conclusion are in the context of a "platform" design approach, with all the efforts concentrated on having a common framework instead of the option to create a new hardware for every new feature needed in production. If this second option is taken, all the study needs to be made again and probably the results can be different.

4.5 Embedded System Design Methodology

After finally selecting which one will be the core technology of the embedded system architecture, the next step in the platform design approach is to define how it has to be managed a project with this kind of architecture.

The possibility to define a correct system design methodology has been one of the core points to select the FPGA + soft-core proposals, because that correct design flow will allow to later reuse all this knowledge in different projects. The first decision that must be taken is to select the main FPGA device. Currently the two market leaders in programmable logic solutions are Altera and Xilinx, with similar product portfolio and design tools. As BCN Tec company already owns design licenses and software tools for Altera devices, it has been decided quickly to develop the architecture based on Altera low-cost FPGAs series, known as Cyclone Series, and his marquee 32-bit RISC microprocessor NiosII

Although the familiarity with the embedded system design methodology is very important to follow this project work, the generic details are a bit out of the scope of the project, so all the generic information is included in the *Annex B: Embedded System Design Methodology* as a reference.

Part 2. Proof-of-Concept prototype

Chapter 5

Proof-of-Concept prototype general architecture

In this chapter the proof-of-concept prototype that will be used to show the feasibility of this project will be introduced. This kind of feasibility studies are usually made on projects with a significant amount of complexity and uncertainty, usually in the early stages of development, in order to get enough know-how to finally take a go/no go decision or make a complete design specification for the next phase besides other objectives.

After the explanation of the proof-of-concept prototype objectives, the chapter follows with the explanation of the functionality that has been selected to show the full potential of the final solution, the HDMI signal generation.

The chapter introduces all the concepts necessary to make the reader familiar with this technology, and after some benchmarking activity, the prototype architecture is explained together with the main decisions on this stage of development. Finally, using all the know-how got from the benchmarking study, a list of targeted features and out of scope features is compiled, in order to be able to better evaluate the success of this proof-of-concept prototype at the final stage of development.

5.1 Proof-of-Concept prototype objectives

In the previous chapters the general objectives of this project have been explained, the core architecture has been already selected, and the methodology to develop all future work has also been defined. Anyway, to finally decide about the long-term feasibility of this project, it has been considered to develop a proof-of-concept prototype, which will be used as a benchmark to prove out the design approach developed in the previous chapter. The key objectives that should be achieved by developing this prototype are:

- Identify potential flaws in core embedded design architecture or key parts selection.
- Show the feasibility to develop homemade smart equipment with some state-of-the-art functionalities needed for test.
- Measure and compare the functionalities of the prototype against some commercial alternatives.
- In a complex project, this prototype is a confidence assessment to make a go/no go decision.
- Get enough know-how to create a complete design specification for the final embedded systems.
- Analyze how all the design methodology phases are applied to the prototype development, to create appropriate milestones for the final embedded systems design.

In order to achieve all these objectives the prototype will target some high-end functionality from the tree list that has been explained in *Chapter 3.1: Introduction of LCD TV main boards testable features*

It is important to remark the difference between a proof-of-concept prototype that is not attempting to be similar to the final boards neither in aspect nor functionality and is not intended to be an early version of a product design, from a functional or working prototype for one of the final embedded boards.

5.2 Proof-of-Concept prototype targeting high-end functionality: HDMI generation

5.2.1 HDMI fundamentals

It is not the aim of this chapter to explain all the complexities of the HDMI standard, anyway a basic introduction of the concepts and technology is made in *Annex C.1: High-Definition Multimedia Interface (HDMI)* that covers the fundamental concepts to understand the HDMI technology and that are needed to follow the subsequent development of the proof-of-concept prototype. These basic concepts are deeply introduced later in the corresponding chapters of this project, but the information in the *Annex C.1* can always be used as a reference.

5.2.2 HDMI block in LCD-TV main boards

In the block diagram shown in Figure 5.1 is represented all the hardware involved in the HDMI block of a high-end LCD-TV, specifically FY2010 AZ1L.

The current trend in the HDMI market is to have at least 4 HDMI inputs, and that creates the necessity of smart HDMI switches because usually the main SoC only have one HDMI receiver core. Usually these smart switches comes with pre-programmed HDCP keys to deal with all the HDCP protocol issues, authentication, encryption and renewability, thus providing an unencrypted TMDS data link between the switch and the main SoC. Also the smart switch acts as the I^2C EEPROM that should be hanging in every I^2C bus, as it has an internal image for every input, eliminating the need of an extra external EEPROM in every bus, that has to be written in production. That kind of integrated solutions allows the manufacturers to focus their efforts on video processing or other added value features, instead of dealing with the complexities of switching high-speed differential signals like TMDS or HDCP protocol.

The HDMI-RX core in the main SoC contains the de-serializer, to demultiplex the incoming unencrypted TMDS data stream into audio and video separate data streams and also the packet stream. Usually this is made by hardware-accelerated processing using the SoC memory system, leaving demultiplexed data stored into memory for further image or audio processing using other SoC capabilities.

As it can be seen, there is also an ARC channel, a buffered digital audio output directly from main SoC. HPD, DDC and +5V signals are managed directly by the switch, and the CEC protocol, managed by the small secondary/standby microcontroller.

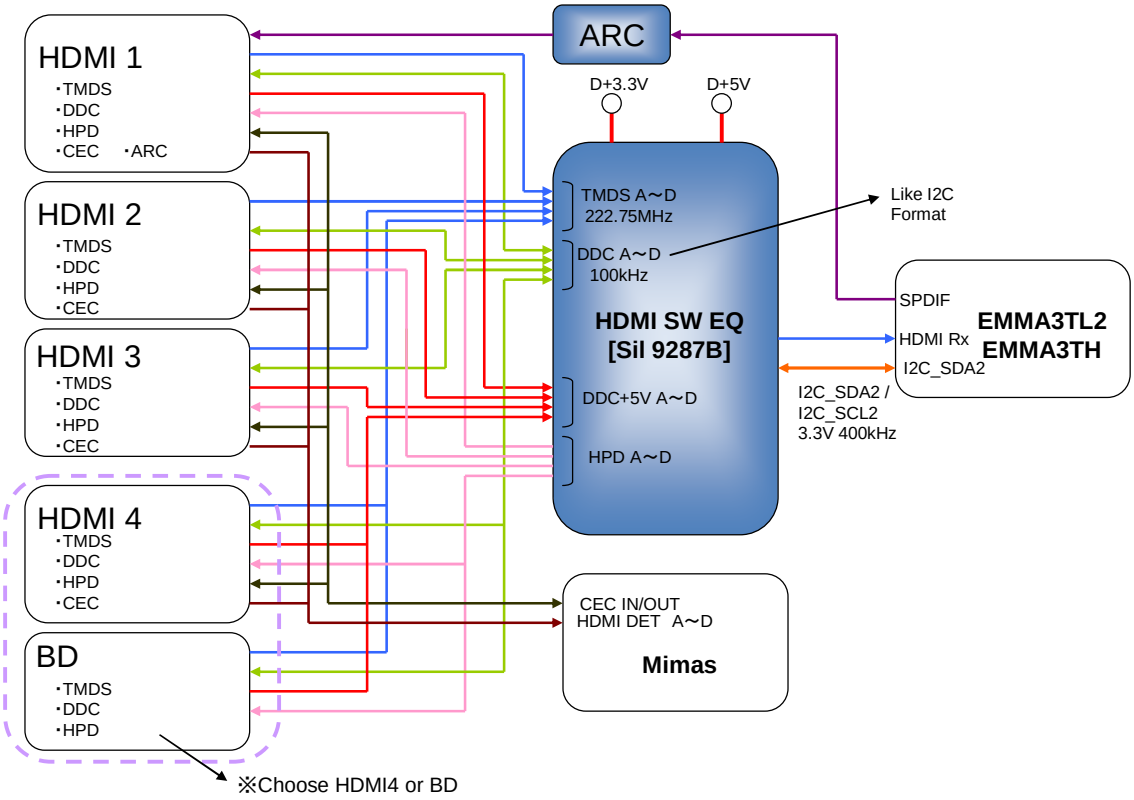
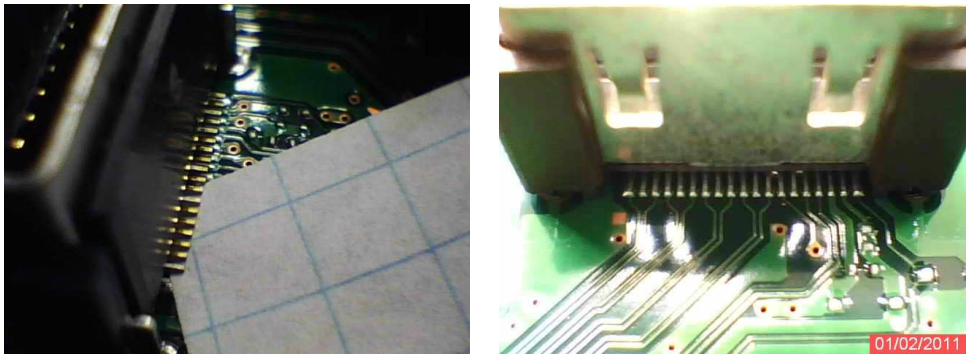


Figure 5.1. Block diagram of HDMI block in high-end LCD-TV

5.2.3 Basic concepts of HDMI testing in LCD-TV

This subsection introduces the HDMI functional test strategies, that has the target to cover all the hardware involved introduced previously.

The empirical experience of some years dealing with HDMI technology says, that typical defects found in HDMI block of LCD-TV boards are problems with the soldering of the HDMI input connector, some examples are shown in Figure 5.2. This connector has a very low-pitch, high-density of pins and to sum it up, the connector case blocks the vertical field of view of solder pads, making impossible to use the AOI to visually inspect the joints. Main problems are produced by solder bridges, no contact produced by a lack of solder paste, bent pins or raised connectors. Other common problems are the defective parts from provider, for both the HDMI smart switches and main SoCs.



(a) Raised connector *(b)* Bridges under cover case

Figure 5.2. Typical HDMI defects because of bad solder in HDMI input connector

In *Annex C.1: High-Definition Multimedia Interface (HDMI) Fundamentals,* all the fundamental concepts of HDMI specification are introduced.
The HDMI functional test is based on testing individually all the HDMI features explained in the *Annex C.1: High-Definition Multimedia Interface (HDMI) Fundamentals,* to find all the previously exposed defects in the produced boards. The test is structured in the following test steps.

HDMI Extended Display Identification Data (EDID) Write/Read Tests
The EDID is a non-volatile memory (NVM) in the HDMI sink that stores the basic information that the HDMI source should know about the sink, like supported audio and video formats, etc. This NVM is connected to the source by a standard I^2C bus channel. These tests are not only made to assure that there is no problem in the hardware involved, especially I^2C bus of every input, but also to write the HDMI EDID data that later is going to be read from the source devices to know the capabilities of the LCD-TV.
In architectures with a smart switch, the EDID is stored internally in the switch, and it has to be written from the main SoC as usually there is no way to write it directly from the HDMI connector. In other architectures with external EEPROM for every HDMI input, the EEPROM has to be written from the connector for every input with its customized data. Later the EEPROM has to be read like any commercial device like DVDs is going to make, to assure data is the correct one. In architectures with a switch with internal EEPROM it is very important to assure that data read from every input is different, as differences of data are managed by the switch and in case of malfunction if received data from every input is always the same it can cause a market problem.

Audio/Video Tests
In these kinds of tests, an HDMI signal is introduced in every input to check input path for TMDS data is correct and there is not a defect in some of the hardware involved. Usually this test is the most costly in test time, as for every input signal, smart switch has to be routed to the selected input in the LCD-TV, HDCP protocol has to be correctly handled and main SoC has to deserialize the TMDS data and process it. That usually means more than 5 seconds until image appears in the output to LCD panel, and audio appears in the audio outputs like speakers or line out. In order to test the Rx core in the main SoC, usually different combinations of video resolution, pixel encoding and color depth are used. In order to make an intensive test and also stress the DDR-SDRAM subsystem in the board, the most restrictive check is usually 1080p with a 12bit color depth and RGB 4:4:4 or YCbCr 4:4:4 encodings.

Consumer Electronics Control (CEC) Tests
In this test, there is not much hardware involved, only shared CEC line of each HDMI input and the I/O pin of the microcontroller that implements the CEC protocol.
There is two ways to test that hardware. First one is based on assuring that line is not stuck to high or low values, by reading logic values received by the microcontroller in the I/O port. The other method consists on sending a CEC command from the source device to the LCD-TV and check that instruction have been received.

Hot-Plug Detection (HPD) Tests
In this test again only one signal is involved, the HPD. Typically source devices trigger the read of source devices EDIDs by detecting a transition in HPD signal. That means the LCD-TV have the ability to force an EDID re-read by pulling HPD line low during some short time. The best way to check there is no defect is to detect that line is not stuck to a fixed logic value,

checking a transition in the source device, when an EDID re-read is triggered from the LCD-TV.

ARC Tests
This test assures that the ARC channel from LCD-TV is sending S/PDIF audio to the source device. Usually S/PDIF audio is send in an optical fiber link, but in the HDMI connector, the signal has the same electric behavior that when a 75Ω coaxial cable is used, that means that a level conversion phase is needed before connecting the signal to a microcontroller with TTL levels. Usually this test is executed by outputting a test audio signal from the LCD-TV that is later received in the 14th pin of the HDMI connector, and sent to S/PDIF RCA input of an AV AMP to check that audio received is the expected one.

5.2.4 HDMI generation commercial alternatives. State of the art.

As it had been explained, the target of this project is to save money in production costs by reducing the investment needed in automatic functional test machines. In previous chapters, a comparison had been made between the cost of the current CBA and the estimation with the new hardware architecture based on homemade test equipment. Anyway, a research has been made in the particular field of HDMI or digital video signals generation, not only to compare the proposed solution with the currently used one, but also to evaluate it against the current state of the art commercial solutions. It is also important to remark that the final embedded systems would target more than one functionality, so this is not a completely fair comparison..

Commercial equipment with HDMI outputs
The first solution that was taken when first HDMI inputs appeared on mass-production LCD TVs, was based on using a commercial DVD with an HDMI output, running continuously a video with an still image pattern and two audio channels reproducing continuously single audio tones of different frequencies.
Advantages:

- Low unit costs, around 50€, and no development needed. Ready-to-use solution.
- Functional test very similar to what final consumer expects from the LCD-TV.

Disadvantages:

- Reliability problems as commercial products are not prepared for 24h/365days use in production environment. Long-term cost increased with substitution of optical units and line stops due to malfunction of equipment. This problem could be solved by using Multimedia Hard Drives with HDMI output.
- No remote control interface to dynamically change between patterns, resolution, audio or video formats during test time. Not ready for multiple resolutions test.
- For HDMI high-end features like 12-bit video output, 3D over HDMI or Audio Return Channel (ARC), the commercial products like Blu-ray set-top-boxes, are not so easy to find and the cost also increases.
- For multiple inputs test with only one source, a full-repeater was needed.
- Not able to run special tests like EDID reading/writing, HDCP testing, HPD or CEC.

This solution was discarded for the long term due to the low flexibility that offers, as equipment has to be configured offline and later is not possible to change the condition during test. For example it was impossible to run tests at different resolutions 720p/1080p or with different patterns. Also the impossibility to run special tests implies that board coverage is not the maximum one, and that is one of the targets of the project.

Analog pattern generator + Composite Video / Audio L/R to HDMI converter

This option is used in current CBAs as there is already an analog pattern generator for analog inputs tests that can be used as video/audio source for the video converter. There is also a big range of video converters, from different analog outputs like VGA, composite video or component video to HDMI. In the particular case of current CBA, a commercial SCART to HDMI converter was used because the pattern generator had a free SCART output. An SCART output has one composite video output and two analog audio channels that are combined in the video converter to generate a 720p HDMI signal.
Advantages:

- Functional test very similar to what final consumer expects from the LCD-TV.
- Reliability increases because the used pattern generator is industrial equipment, and video converters do not have any mechanical parts like the optical units in DVDs.
- Remote control interface to change the video signal pattern.

Disadvantages:

- Unit costs increase, around 3k€, because of the use of an industrial pattern generator. Although the pattern generator currently used in CBA is not in the high-end series, and do not have too much functionalities, the price is around 3k€. The video converter is by far cheaper, around 100€.
- Although it is possible to change the video pattern, there is no remote control interface to dynamically change between resolutions, audio or video formats during test time in the video converter. The current video converter used has a fixed output resolution of 720p and is not useful for multiple resolutions test.
- No support for HDMI high-end features like 12-bit video output, 3D over HDMI or Audio Return Channel (ARC).
- For multiple inputs test with only one source, a full-repeater was needed.
- Not able to run special tests like EDID reading/writing, HDCP testing, HPD or CEC.

This solution again was discarded for the long term due to the low flexibility that offers, as equipment only have one configuration and later is not possible to change the condition during test. For example it was impossible to run tests at 1080p. Also the impossibility to run special tests implies that board coverage is not the maximum one, and that is one of the targets of the project.

High end pattern generator with HDMI output
As HDMI technology had reached maturity and is an ever present in all the LCD-TVs, the main pattern generator manufacturers had updated his high-end products to add HDMI outputs to their products. AstroDesign is the manufacturer of some of the pattern generators currently used in BCN Tec LCD-TV assembly lines, and his VG-489C model has an HDMI output that supports HDMI specification 1.3a and is optimal for HDMI testing.
Advantages:

- Industrial equipment with high reliability.
- Remote control interface to change all the configurable parameters of the HDMI generator, patterns, video formats and resolutions, audio settings, etc.
- Support for some HDMI high-end features present in HDMI 1.3 specification like 12-bit video output.
- Possibility to run special tests like EDID reading/writing, HDCP testing, HPD or CEC.

Disadvantages:

- High unit costs as standard price for VG-849C is around 9k€
- No support for new features present in HDMI 1.4 specification like 3D over HDMI or Audio Return Channel (ARC).
- For multiple inputs test with only one source, a full-repeater was needed.

This solution although is the optimal one on terms of performance, has been discarded as the investment needed due to the high unit price of that pattern generator, goes completely against the main objective of the project, that is to reduce the total costs of hardware for each CBA position. Anyway, the impressive list of features can be used as a benchmark to compare the capabilities of the proposed solution.

5.3 Proof-of-concept prototype architecture design

5.3.1 Introduction

Previously in this chapter the objectives of the proof-of-concept prototype have been explained, and the main technologies involved around it had been introduced.
The specific details of the prototype architecture are going to be described in the following points.

5.3.2 Embedded system block

In the *Chapter 4: Embedded system architecture selection,* it has been explained why the system will be based on an FPGA with a soft-core microprocessor inside, and the advantages from that embedded architecture have been introduced.
In order to have the hardware platform for the proof-of-concept prototype, several options have been evaluated, but in order to speed up the development time needed, it has been decided to use an off-the-shelf Altera development kit instead of making the complete hardware system from scratch in this early phase of development. PCB design for large systems with high pin-count FPGAs, external memory interfaces like DDR-SDRAM or Flash and communication interfaces like Ethernet mean a lot of development time with CAD tools, only to create the platform where the system will be running, and there is a high risk of PCB design flaws that will dangerously increase time-to-prototype.
As in BCN Tec there are some spare Altera development kits from older projects, one of them that have the desired features has been selected to develop the prototype. The Nios Development Board, Cyclone II Edition (DK-NIOS-2C35N), Nios 2C35 development kit from now on, provides a hardware platform for developing embedded systems based on the Altera® Cyclone II devices. The block diagram of the Nios 2C35 development kit is shown in Figure 5.3.
Among others, the kit provides the following features that will be used in the prototype:

- Cyclone II EP2C35F672C6N FPGA with 33,216 logic elements (LE) and 483,840 bits of on-chip memory
- 16 MBytes of Flash memory
- 2 MBytes of synchronous SRAM
- 32 MBytes of double data rate (DDR) SDRAM
- On-board logic for configuring the FPGA from flash memory
- On-board Ethernet MAC/PHY device and RJ45 connector
- Two 5.0 V-tolerant expansion/prototype headers each with access to 41 FPGA user I/O pins
- Four push-button switches connected to FPGA user I/O pins
- Eight LEDs connected to FPGA user I/O pins
- JTAG connectors to Altera devices via Altera download cables
- 50 MHz oscillator and zero-skew clock distribution circuitry
- Power-on reset circuitry

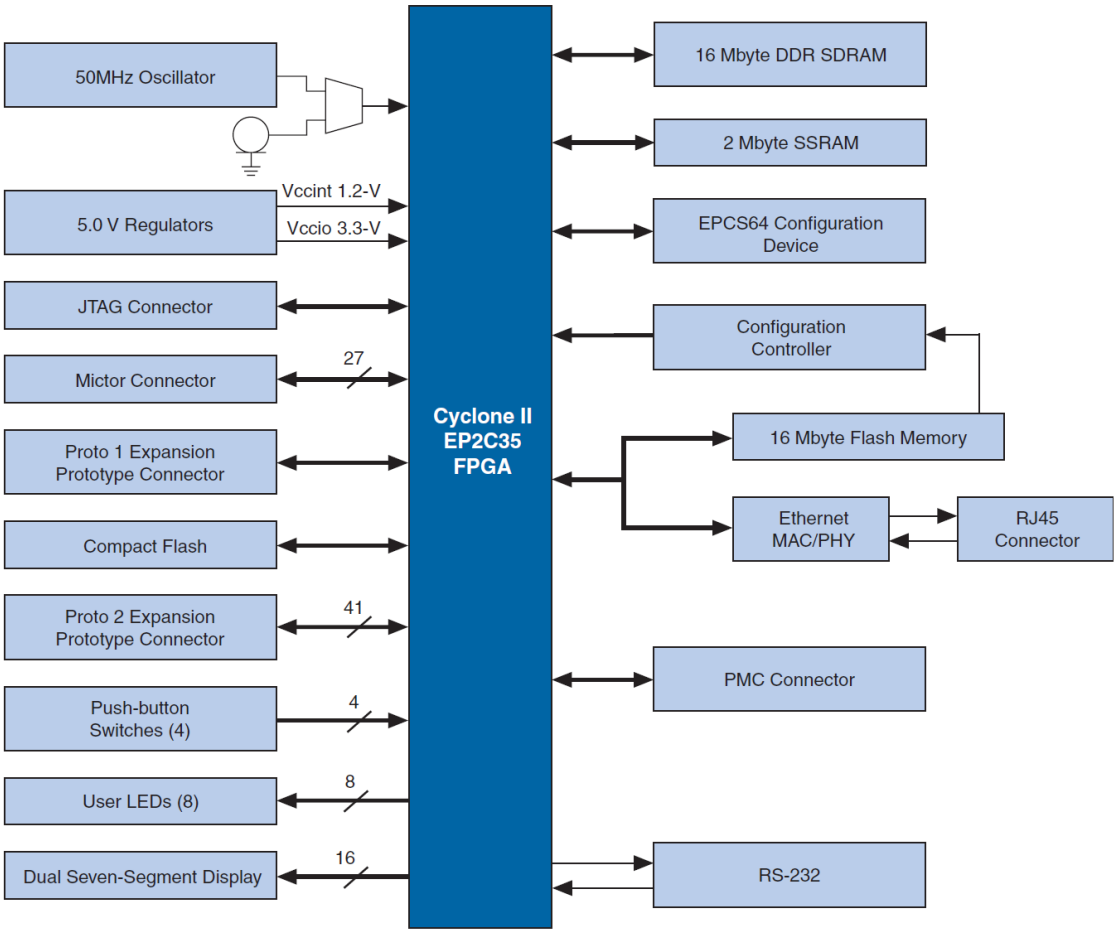


Figure 5.3. Nios II Development Board Cyclone II Edition Block Diagram

As it can be seen from the list of features, that 2C35 development kit fulfills all the requirements that where remarked in *Chapter 4: Embedded system architecture selection*. The Ethernet interface will be used as main communication interface to remotely control the equipment and download application data and settings, and the two expansion prototype blocks, each one with 41 I/O pins, 3.3V and 5V reference power rails, and dedicated clock inputs/outputs, will be used to connect a daughter card with all the HDMI generation specific hardware.

5.3.3 HDMI generation block

Once the system controller has been decided, the next thing to do with the prototype is to find a solution for the HDMI generation.

Although Cyclone FPGA devices have dedicated differential input/output buffers on I/O banks that support LVDS, and the serialization of data sent in the TMDS lines potentially can be made inside the device, this idea was discarded because of the high complexities of HDMI specification, and the extended development time needed to deal with all HDMI compliance issues. Another alternative found in the market are the combination of HDMI IP cores with external PHY devices, which can be incorporated quickly in a FPGA prototype or an ASIC. Anyway the market leaders in HDMI cores, Silicon Image™, only sells licenses of these cores to major resellers that can split the investment by making a large production of ASICs, so this solution was also quickly discarded.

Although HDMI technology is still relatively new, it is extended enough to find in the market complete solutions addressed to HDMI source devices. These integrated chips are called HDMI transmitters and not only deal with the physical link issues, but also provide useful data and control interfaces with the main controller to simplify the whole process of the HDMI signal generation.
A list of desirable features is made in order to narrow down the list of potential devices:

- Compliance with HDMI 1.4 specification. Only modern HDMI transmitter devices are already prepared for new features, although some devices can be compatible in some points.
- TMDS clock up to 225MHz. This feature is needed to enable 12-bit video at 1080p@60Hz resolution. Typical HDMI 1.3 compatible devices are only capable of reaching 165 MHz, and that narrows the list of video formats that can be sent by the generator.
- Support for the most common standard video input formats. This video input formats are defined in the CEA-861-E specification that establishes protocols, requirements, and recommendations for the utilization of uncompressed digital video interfaces.
- Support for the most common standard audio transport protocols, like I^2S or S/PDIF. This feature is required to quickly interface with the main controller system in charge of generating the audio stream.
- Compatibility with HDCP 1.4. The possibility to encrypt the TMDS output is important as the HDMI sinks need to test that its own HDCP keys are correctly written. Optionally if the transmitter device come with pre-programmed HDCP keys it would also be perfect to accelerate development time, as the process to obtain a license and get keys for source devices is long and complicated.
- (Optional) Support for ARC. As new HDMI sinks already have the Audio Return Channel feature, an automatic buffering to standard TTL S/PDIF interface would be a nice feature to have to later interface that signal directly to the main system controller.
- (Optional) Integrated CEC buffer/controller. If HDMI transmitter already has this functionality it will accelerate the development time, offloading the responsibility from main controller to the transmitter itself, and without dealing with the complexities of CEC protocol in the main controller.
- (Optional) Support for the most common 3D video formats. This feature is required to avoid a quick obsolescence in a quickly-changing market, as more an more HDMI sinks will incorporate 3D support, and the ability to test it would be required in a near future. Again, the specific 3D video formats are defined in CEA-861-E specification.

Some research has been made between the vast range of HDMI transmitters solutions available to select the device. The different options are listed in Table 5.1.

IC Name	Manufacturer	Price	Comments
AD9389B	Analog Devices	~6€	**Advantages:** NDA signed with AD and available development information. **Disadvantages:** HDMI 1.3 device with no advanced features. Only 8-bit video support, maximum TDMS clock up to 165Mhz. External EEPROM needed for HDCP keys.
ADV7511	Analog Devices	~10€	**Advantages:** NDA signed with AD and available development

			information. HDMI 1.4 device with advanced features, ARC and 3D video. 12-bit video support, maximum TDMS clock up to 225Mhz. Integrated CEC controller. Pre-programmed HDCP keys. **Disadvantages:** Availability issues.
SiI9334	Silicon Image	~15€	**Advantages:** NDA signed with AD and available development information. HDMI 1.4 device with advanced features, ARC and 3D video. 12-bit video support, maximum TDMS clock up to 225Mhz. Integrated CEC controller. Pre-programmed HDCP keys. **Disadvantages:** NDA not signed with Silicon Image and development information is not available yet. Availability issues.
TDA9981A	NXP	~4€	**Advantages:** NDA signed with AD and available development information. HDMI 1.4 device with advanced features, ARC and 3D video. Pre-programmed HDCP keys. **Disadvantages:** Only 8-bit video support, maximum TDMS clock up to 165Mhz. Non-integrated CEC controller. NDA not signed with NXP and development information is not available yet.

Table 5.1. HDMI transmitter commercial devices

The limiting point that has been found during the research is the lack of available documentation for development. As HDMI transmitter is still a very raw market with a long way to go, and new devices with extra features are constantly appearing, companies do not offer the needed development information without signing before a Non-Disclosure Agreement (NDA), that basically is a legal agreement between two parties to share confidential material, knowledge or information, thus restricting access to other third-party companies.

From technical point of view, the AD9389B has been quickly discarded considering that it is a nearly-obsolete device because it do not offer any of the new features present in HDMI 1.4 specification and is also limited to 8-bit video. The NXP device has been considered because potentially the relationship between BCN Tec and NXP could accelerate development time and solve availability issues, but is also well behind the ADV7511 and SiI9334 alternatives as it only has an 8-bit input video interface and other optional features are not supported. ADV7511 and SiI9334 devices are both located in the high-end segment of HDMI transmitters with all the required features and also optional nice-to-have features like ARC buffering, CEC controller or pre-programmed keys. In terms of control, both have I^2C

interface to the main system controller that can change the internal registers to configure the device.

The process to sign an NDA between BCN Tec and Analog Devices (AD) to share information was long and tedious, but it was made some time before the start of the proof-of concept prototype development. The initial communication problems found when contacting with Silicon Image (SiI) finally made the decision of selecting ADV7511 easier to take. All the devices had some availability problems in the long-term, but in the case of the ADV7511 devices there were also engineering samples of the device quickly available for the engineering prototype development, making it the common sense decision not only for the available documentation and the detailed advantages from technical point of view, but also for the possibility to make the time-to-prototype shorter, something that would help the long-term feasibility of the project.

5.3.4 Detailed system architecture. Block Diagram

In the following diagram the architecture of the complete system can be appreciated. As a summary, the FPGA hardware platform will be configured as a NiosII system with several cores controllable from the microprocessor. The HDMI transmitter in the external HDMI transmitter daughter card will receive the uncompressed digital video signals that will be generated on the FPGA in a core controlled by the NiosII processor, and also the digital audio in I^2S format that will be generated internally in another FPGA core controlled by NiosII. An I^2C Master core will provide the communication from NiosII controller to the HDMI generator. The NiosII firmware will be loaded and booted from the main flash that can also be used for storage, and the code will run in the DDR SDRAM or SSRAM.

The LAN91C111 Ethernet MAC/PHY present in the 2C35 board can be used to connect to Ethernet networks, and ethernet will be used as the main remote control interface between the host PC and the equipment, with a client (host PC) – server (main board) architecture.
Also a couple of analog audio to I^2S A/D converters will be integrated in the HDMI generator board, in order to have a backup if there are problems in the I^2S generation core development. In this case, these A/D converters can act as a backup solution to fully test the audio capabilities of the HDMI transmitter. Finally the NiosII system will use the on-board oscillator in the 2C35 as the master clock, but the HDMI transmitter board will have two additional master clocks for audio/video to solve the impossibility to generate all the different frequencies from the 50Mhz clock. In the following chapters the information about all these points and development decisions will be extended.

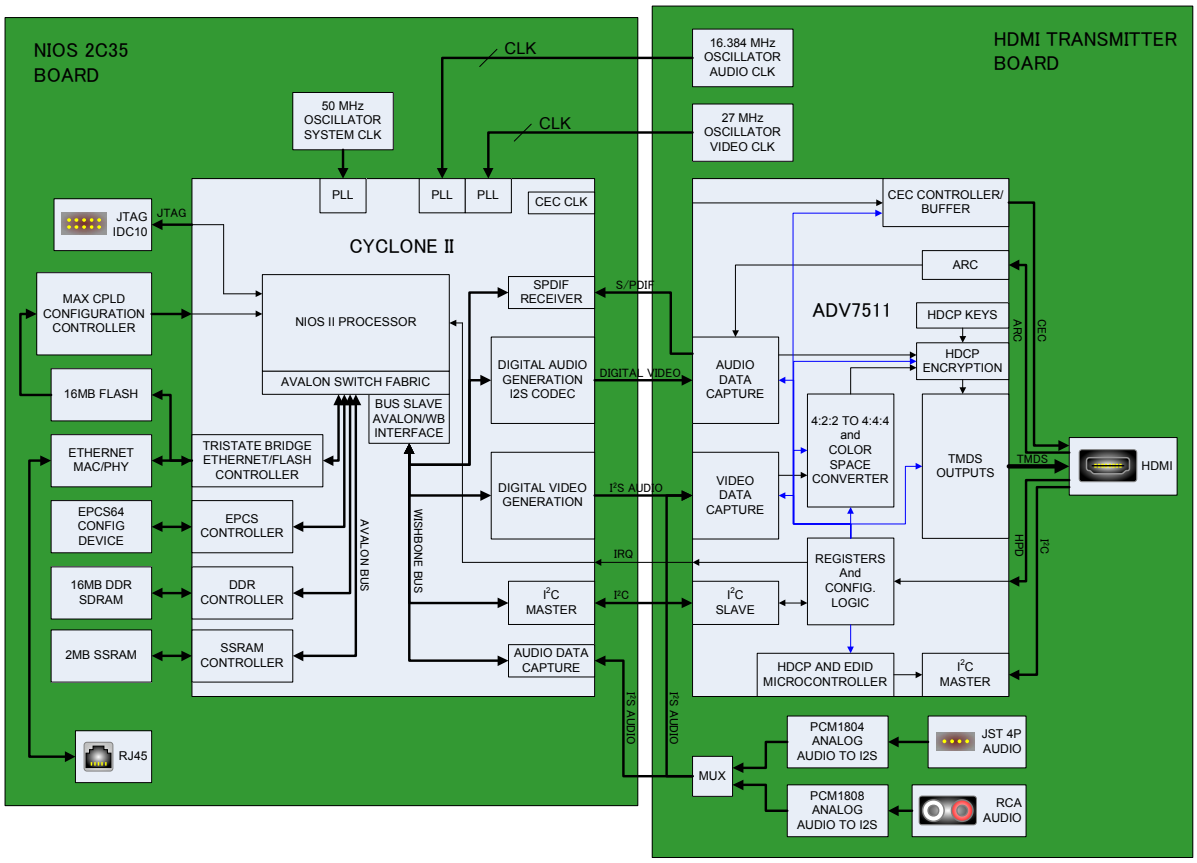


Figure 5.4. Proof-of-Concept prototype complete system block diagram

5.4 HDMI generator board scope

This is a high-level description of the features and functions that would characterize the product, in this case the proof-of-concept prototype of the CBA Low-Cost new hardware architecture, called HDMI generator board. This name is used from now on to talk about the proof-of-concept prototype.

Target features:

- Support for DVI and HDMI modes.
- Support for all typical video output formats according to CEA-861-E specification. Ability to dynamically between different video output formats.
- Possibility to create custom video formats tweaking the generation of the video timings.
- Possibility to change the aspect ratio information embedded in HDMI signal.
- Support for deep color mode, color depth selection from 8-bit to 12-bit per component.
- Basic test patterns included and selectable
- Support for HDMI audio at multiple sample rates.
- Possibility to change audio output frequency range independently for every channel and with a good frequency resolution.
- Special control mode functions like audio input mute or AV mute over HDMI.
- EDID read support.
- HPD monitoring and control.
- Low-power modes supported when HDMI generator is not connected to a sink.
- Support for different InfoFrame packets generation.

- Ethernet remote control interface to select among all supported configurations.

Out-of-scope features:

- Support for 20 or 24 bits audio samples depth. Limited to 16 bits per sample.
- Support for different audio waveforms like frequency sweeps. Limited to sine waves.
- Support for different color spaces and color component subsampling like YCbCr 4:4:4 or YCbCr 4:2:2. Limited to RGB 4:4:4, color space converter no used.
- HDCP digital content protection. Video and audio are always sent unencrypted.
- ARC audio return channel analysis. Supported but not implemented.
- Generation of special 3D video formats. Supported but not implemented.
- CEC controller functions. Supported but not implemented.

Basically in the out-of-scope features list there are only features that are not supported by the current solution used for HDMI generation in BCN Tec production. As the main objective is to demonstrate the feasibility of the new hardware architecture in front of the old one, based on commercial equipment, it has been decided to implement in the first phase only the strictly necessary to make a straight swap replacement of the current solution. This approach allows testing the prototype in production, in order to estimate the long-term reliability of the newly proposed architecture. In a second phase, some of the other features that could be also useful in production, like HDCP, ARC or CEC, could be implemented to show the full potential of the proposed solution and give some value added features in comparison with current scenario.

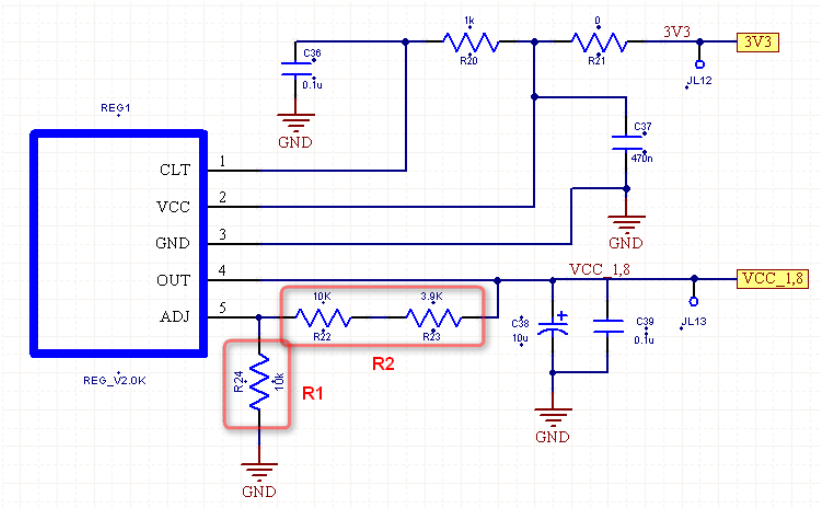
Chapter 6

Schematic & PCB design of HDMI transmitter board

The 2C35 development kit is a ready-to use platform that does not need any hardware change, and that have some expansion connectors to expand the features of the system with daughter cards. This chapter describes the main decisions about the hardware design of the HDMI transmitter daughter board that is part of the proof-of-concept prototype. The interaction between the 2C35 development kit and the HDMI transmitter board is also introduced during this chapter.

6.1 Power

The 2C35 board has on-board circuitry to generate several power supply lines, some of them are internal like the 2.5V for the DDR and the 1.2V for the CycloneII, but other can be used for the daughter cards.
In the PROTO1 & PROTO2 expansion connectors there are available two power lines, +5V and +3.3V that is also the power source of all the FPGA I/O pins. As these power supply lines are available in the PROTO connectors, the HDMI generator board does not have any other external power supply input.
The ADV7511 HDMI transmitter requires a +1.8V supply for the main operation an adjustable linear regulator (LDO) with low dropout, BD00KA5WF, is used with a resistor network to create the main supply for the ADV7511.
The 500mA maximum output current of BD00KA5WF is enough for the ADV71511 that according the datasheet has a maximum consumption around 180mA in worst state when working with full capabilities active.



$$Vadj = 0.75V(Typ)$$

$$Vout = Vadj \cdot \frac{R_1 + R_2}{R_2}$$

$$Vout = 0{,}75 \cdot \frac{10k + 10k + 3k9}{10k}$$

$$Vout = 1{,}7925V$$

Figure 6.1. Output voltage configuration for BD00KA5WF LDO adjustable voltage regulator

ADV7511 I/O lines can operate at both +1.8V and +3.3V signal levels. In this case, to gently interconnect with the FPGA I/O pins, the power supply pin to select I/O voltage is connected

at +3.3V, and the pull-ups of the ADV7511 I^2C internal bus that is connected to the FPGA are also connected to +3.3V.
For the other I^2C bus connected to the HDMI connector, the pull-ups are connected to +5V as this is common on HDMI devices, and according the datasheet I/O pins of ADV7511 are tolerant to +5V logic levels.
The +5V are also used in the HDMI connector, as HDMI sources always send a +5V line to sink devices. A simple serial resistor protection is used to not damage the board in case the +5V line is shorted in a malfunctioning HDMI sink device.
In the stereo audio A/D converters the +5V are used as analog power supply, and the +3.3V are used as digital power supply, as again all these lines are connected to FPGA I/O pins that also operate at +3.3V.
As recommended in the ADV7511 datasheet the +1.8V have been separated in three different power supply domains, analog domain for TMDS outputs, digital domain for digital logic and I/O pins and clock domain for clock PLL. Each domain has his corresponding noise-filtering LC filter to attenuate noise present in the linear regulator output, and all power pins also have their corresponding bypass capacitors, as it can be seen in the schematics.

6.2 Clocks

The 2C35 development board already includes a 50MHz oscillator that is routed directly to main Cyclone II device, and to the PROTO connectors. Cyclone II EP2C35 devices have up to 16 dedicated clock input pins capable of driving the 16 global clock networks.
The 50MHz system clock is going to feed two of the four PLL in the Cyclone II device, and the different coherent internal clocks synthesized will be used to drive different parts of the 2C35 board, like the DDR, SSRAM, Flash, and also internally the NiosII processors and the different VHDL cores.
Anyway, a 50MHz clock is not the best selection to generate video clocks, as PLLs can not synthesize all the frequencies needed in a multi-resolution video generator. Also the same reasoning can be applied to the audio clocks, as the Cyclone II PLL is not able to generate all the frequencies needed in an I^2S system capable to generate audio at different sampling rates.
Eventually, from a hardware point of view it is finally decided to use two additional clock oscillators in the HDMI transmitter daughter card, a 27MHz oscillator for the video subsystem, and a 16.384MHz oscillator for the audio subsystem. In the PROTO connectors, from the Cyclone II point of view there are two input clocks, one for each block, proto1_PLLCLK and proto2_PLLCLK, that are used to feed the FPGA with audio and video clocks.
All these issues and a complete explanation of the reasons behind the final clock scheme are further expanded in the *Chapter 8.2 - Clock network design.*
There are also two clock outputs, which are used to send the video pixel clock and I^2S audio reference clock to the HDMI transmitter clock inputs. These two clocks have also been routed to two common I/O pins as well as two dedicated pins, to test the possibility to not drive the HDMI transmitter clocks from dedicated outputs. Although there is no real necessity in the prototype, maybe this option can be mandatory in some future design with less dedicated clock outputs available. The two options are selectable depending on some resistors mount.

6.3 Video Block

The ADV7511 accept video data from as few as eight pins, to as many as 36 pins. It also has additional pins for HSYNC, VSYNC and DE.

It has been decided to prepare the hardware for every possible scenario about bits per color, format and syncs type, in order to give maximum flexibility. In this case the worst scenario is 12-bit per color RGB4:4:4 with separate syncs.
There are enough I/O pins in the PROTO connectors for the 36 data signals, and as bit ordering can be internally dealt with the FPGA, the assignments have been made trying to ease the routing in the PCB, keeping all the signal traces with similar lengths. Also the HSYNC, VSYNC and DE signals have been connected directly to PROTO I/O pins. As explained previously the pixel clock to the ADV7511 can be controlled from a dedicate output clock pin or a common FPGA I/O pin.

6.4 Audio Block

The ADV7511 has multiple audio input interfaces, and is capable of receiving audio data in several different formats.

- Inter IC Sound (I^2S)
- Sony/Phillips Digital Interface (S/PDIF)
- Direct Stream Digital Audio (DSD)
- Direct Stream Transfer (DST)
- High Bit-Rate (HBR)

As in the Video Block it has been decided to route all the 15 pins of the different audio interfaces to I/O pins in the PROTO connectors, in order to give a total flexibility about which audio format and interface is used to transmit audio data from the Cyclone II to the ADV7511.

6.5 TMDS outputs

The three TMDS data channels and the TMDS clock channel are connected from the ADV7511 to the HDMI connector. Some special considerations for routing these high-speed data lines are explained in *Section 6.12: PCB special design considerations*
In order to reduce the EMI emissions at higher frequencies, common mode chokes have been placed in the TMDS lines as close to the ADV7511 as possible.

6.6 I2C buses

The ADV7511 has two different I^2C buses, an external one in which the ADV7511 is the master controller to read the EDID in the HDMI sink device connected to the HDMI transmitter board, and an internal one to access the ADV7511 registers from the controller. In this one, the ADV7511 acts as a slave fore receiving and transmitting data over the bus. Optional pull-up resistors connected at +5V have been added in SDA and SCL lines of the bus where ADV7511 acts as a master, because although HDMI specification is not clear in which value is recommended, this one typically will be very low, as maximum capacitance allowed of 50pF in the HDMI source is very strict and this is possibly the best way to achieve quick I^2C rise times without using extra buffers or I^2C accelerators. Anyway is good to remark that the main limiting point in the rise time is always going to be the cable connecting the source and sink device.
For the internal I^2C bus as there are no extra requirements, typical pull-ups of 4k7 at +3.3V have been selected.

6.7 HPD

The HPD pin of ADV7511 that detects the presence of an HDMI sink device is directly connected to the HDMI connector as is tolerant to +5V logic levels.
A pull-down resistor have been added in the HPD line to force the not connected state, in order to reliably differentiate between a floating state, when HDMI sink is not connected and a high-level state when the sink is connected.

6.8 CEC

For the CEC pin, a 27k pull-up resistor at +3.3V is used to lift the bus voltage to +3.3V when bus is idle, as CEC is like the I^2C an open-collector bus. As bit rates in CEC bus are very low compared with modern bus standards, around 500bps, rise and fall times requirements are more relaxed, and this is the reason that pull-up resistor is so high. Internally ADV7511 needs a valid clock in CEC_CLK input to later generate the timings in CEC bus, and a common I/O pin in the PROTO connector is reserved for this matter, as the clock can have any value between 1 and 100MHz, and can be easily generated inside the CycloneII device.

6.9 ARC

The ADV7511's ARC receiver accepts single-ended or common mode signals and directs the resulting S/PDIF signal to S/PDIF_OUT pin. Although the first TV models that have introduced this feature send a single-ended ARC signal, as this is still not a mature technology, the board is left prepared for any mode, routing the HEAC+, unused pin 14, and HEAC-, pin 19 shared with HPD functionality, as differential 100Ω impedance differential signals, decoupled with a 1μF capacitor and each one terminated with a 50Ω resistor to +1.8V. In this configuration the ARC receiver can work transparently with a single-ended and a common mode audio return channel coming from HDMI sink.

6.10 Analog Audio Input Block

This block has been introduced in the HDMI transmitter board as a backup in case of problems with the I^2S Audio Generation core.
Two different commercial stereo A/D converters are used, the PCM1804 and PCM1808. Basically these two converters are included in the low-cost family of converters, as neither of them has a control interface to configure the different features they have. For the basic configuration only input pins are needed, and it has been decided to not force these values and instead of this route this signals to a 8-pin micro-switch, to manually configure it. The main differences between them are the supported sampling frequency that is higher in the PCM1804, and the analog audio input interface, as PCM1808 only accepts single-ended audio and PCM1804 has a differential analog voltage input. Another important thing to remark is that these chips are selected also to explore future applications of audio capturing. In that case PCM1804, although bigger and expensive, it also has the important feature of the differential audio input, useful for testing final Class-D amplifiers stages that drive speakers, a typical case in LCD-TV. This topic is not further expanded because eventually this block has not been used in the project.

6.11 Interconnection

All the interconnections between all the blocks in the HDMI transmitter board and the 2C35 board are listed in the *Annex E - Pinout of interconnection between 2C35 Kit HDMI Transmitter board*. As it can be appreciated nearly all the I/O pins available have been used for this development, and the pin assignment have been made with the PCB routing in mind, because the pin assignments change in the FPGA is favored in front of complicated routing paths.

6.12 PCB special design considerations

The HDMI transmitter board has some of the classic electrical restrictions for high-speed boards. For cost restriction the PCB uses only 2 layers, although for these kind of boards with high-speed data lines and different power domains at least a 4 layers PCB is recommended.

In this case the TMDS signals from the ADV7511 HDMI transmitter to the HDMI connector can run up to 2.25GHZ, so the following restrictions have been applied.

- Length of the traces has been matched to minimize intra-pair skew between the two differential lines (D+ and D-), and inter-pair skew between TMDS channels.
- Differential pairs have been traced parallel to each other
- PCB traces have been designed as far as possible, with a controlled differential impedance of 100Ω.
- The HDMI connector has been put as close as possible to the HDMI transmitter to minimize the parasite resistance path.
- Other parts of the design have been physically separated four times the width of the TMDS lines as far as possible, although sometimes due to the same ADV7511 pinout it is practically impossible
- The bottom layer in the PCB has a ground plane just below the TMDS lines routed in top layer.

The video and audio signals have also been routed trying to minimize its length as much as possible. The 100Ω serial resistor arrays in the data lines close to the ADV7511 have been put to minimize impedance mismatch.
Finally to help the power supply filtering, the recommendation in the datasheet of separate PCB power planes for each of the three +1.8V power domains, have been followed as much as possible considering the PCB uses only two layers, and it is difficult to make separate power planes with a big surface area, that are usually made in the inner layers.

Chapter 7

System Architecture

As it has been pointed out before, the selection of a development kit as main hardware platform speeds up the whole process to launch a functional prototype, not only in terms of cutting down the hardware development time, but also speeding up the system architecture design, as a lot of literature and several functional examples of Nios II based systems using all of board features come with the kit. The available examples, using all hardware features of the platform, simplify all the embedded design related tasks that are covered in detail in *Annex B: Embedded System Design Methodology*, allowing the possibility to focus the development effort in the new functionalities that the prototype is targeting.

This chapter explains how the system has been designed from a functional example, briefly explaining the main system components and their use in the prototype, and focusing above all on the new particular features added, especially to connect other external cores outside of the SOPC Builder system.

7.1 System design using SOPC Builder

As it has been pointed out before, the selection of a development kit as main hardware platform speeds up the whole process to launch a functional prototype, not only in terms of cutting down the hardware development time, but also speeding up the system architecture design.

The 2C35 board comes with several functional examples of Nios II based systems using all of board features. The available examples, covering all hardware features of the platform, simplify all the embedded design related tasks that have been explained in Chapter 4, allowing the possibility to focus the development effort in the new functionalities that the prototype is targeting.

The system design has been started from a partially completed SOPC Builder system that covers all the main hardware features of the Nios 2C35 board. Some components have been removed or modified, and there are some handy additions to be able to control external modules, anyway as the system has not been created from scratch, only a short overview of the components selected is made in Table 7.1, explaining briefly the function for which the component has been selected, to later focus on how the system is planned to work.

Component	Instance Name	Component role in the System
Nios II Processor	cpu	Nios II processor is the main component of the system, acting as the main master controller of most of the other subsystems. It has been selected the Nios II/f core that is the one with the best performance.
DMA	dma	The DMA controller with Avalon Interface is used to offload costly memory transfer tasks from the CPU to the , allowing the CPU to perform other tasks in parallel.
On-Chip Memory	tightly_coupled_ data_memory	Block of fast on-chip memory that is used as a data cache, connected directly to a CPU master.
On-Chip	tightly_coupled_	Block of fast on-chip memory that is used as an instruction cache,

Memory	instruction_ memory	connected directly to a CPU master.
DDR SDRAM Controller MegaCore Function	ddr_sdram	The DDR SDRAM controller handles the complex aspects of using DDR, initialize the device, manage banks, and refreshing at appropiate intervals. The controller also translates read and write requests from local interface to the necessary DDR signals.
Avalon Tristate Bridge	ext_ram_bus	The Avalon memory-mapped tri-state interface allows the Avalon master in the CPU to drive tristate off-chip devices. In the current Nios 2C35 board two tristate off-chip devices like the Ethernet MAC Controller and External Flash share the same bus, data and address pins, and the Avalon-MM tristate interface is the needed interface to use that devices from the main CPU transparently.
LAN91C111 Interface (Ethernet)	lan91c111	The Ethernet controller in Nios 2C35 board comes with a SOPC Builder-ready component with integrated device drivers that could be later used in the Nios II IDEs, allowing the developer to focus in application software instead of low-level hardware details.
Flash Memory (Common Flash Interface)	ext_flash	The common flash interface controller core allows to easily interconnect the main system with an external flash memory compliant with the CFI specification. The flash memory will be used to save persistent settings and also the main application software that Nios II cpu will run.
Avalon Tristate Bridge	ssram_bus	Although the SSRAM bus is not shared with any other device, the SSRAM is a tri-state device that cannot be connected directly to the CPU master port.
Cypress CY7C1380C SSRAM	ssram	The SSRAM interface component controls timings in read/write transactions to the external Cypress SSRAM that is present in the Nios 2C35 board. It needs to be connected always to the master por of an Avalon Tristate Bridge
EPCS Serial Flash Controller	epcs_controller	The EPCS Serial flash controller core allows the system to acces a serial configuration device from the Nios II application software. Usually that devices only store the FPGA configuration memory, but all the rest of space can be used as a general-purpose memory to store non-volatile data.
System ID	sysid	Allows the Nios II IDE to veify that the software is built for the correct hardware version, changing the ID every time a new Nios II system is generated.
Interval Timer	high_res_timer	The interval timer core is used to perform periodic interrupts from the main application software run in the the Nios II system. The timeout period of high-res timer is configured to 1 ms
Interval Timer	sys_clk_timer	The interval timer core is used to perform periodic interrupts from the main application software run in the the Nios II system. The timeout period of high-res timer is configured to 10 ms
JTAG UART	jtag_uart	Enable debugging from Nios II IDE using an USB-Blaster cable acessing to the jtag debug serial port
UART (RS-232 Serial Port)	uart	The UART core is a method to communicate with other subsystems with a serial interface. It will only be used for debugging issues when a USB-Blaster cable is not available.
PLL (Phase-Locked Loop)	pll	The PLL core is used to access to the dedicated on-chip PLL circuitry in the Cyclone II FPGA. This PLL will generate the system clock of 85MHz used in the NiosII and most of the Avalon master/slaves present in the system as well as the ddr and ssram phase-shifted clocks.
PLL (Phase-Locked Loop)	pllsysx2	The PLL core is used to access to the dedicated on-chip PLL circuitry in the Cyclone II FPGA. This PLL is basically used to

		generate a clock half the speed of the system clock that will be used in all the cores external to the main system.
Avalon Interface	bus1	The Avalon Interface is an interface core created to create other Avalon slave cores created outside of the Nios II SOPC Builder system.

Table 7.1. List of selected system components

All the components selected have several configuration parameters; anyway, it is not worth to mention all of them as it is outside of the scope of this work.

When all components are already selected the main interface of the SOPC Builder is used to graphically make the interconnections between components as shown in Figure 7.1. As that system has been created based on one example, the input clocks for every component, the memory-map base addresses and the interrupt priorities has not been modified.

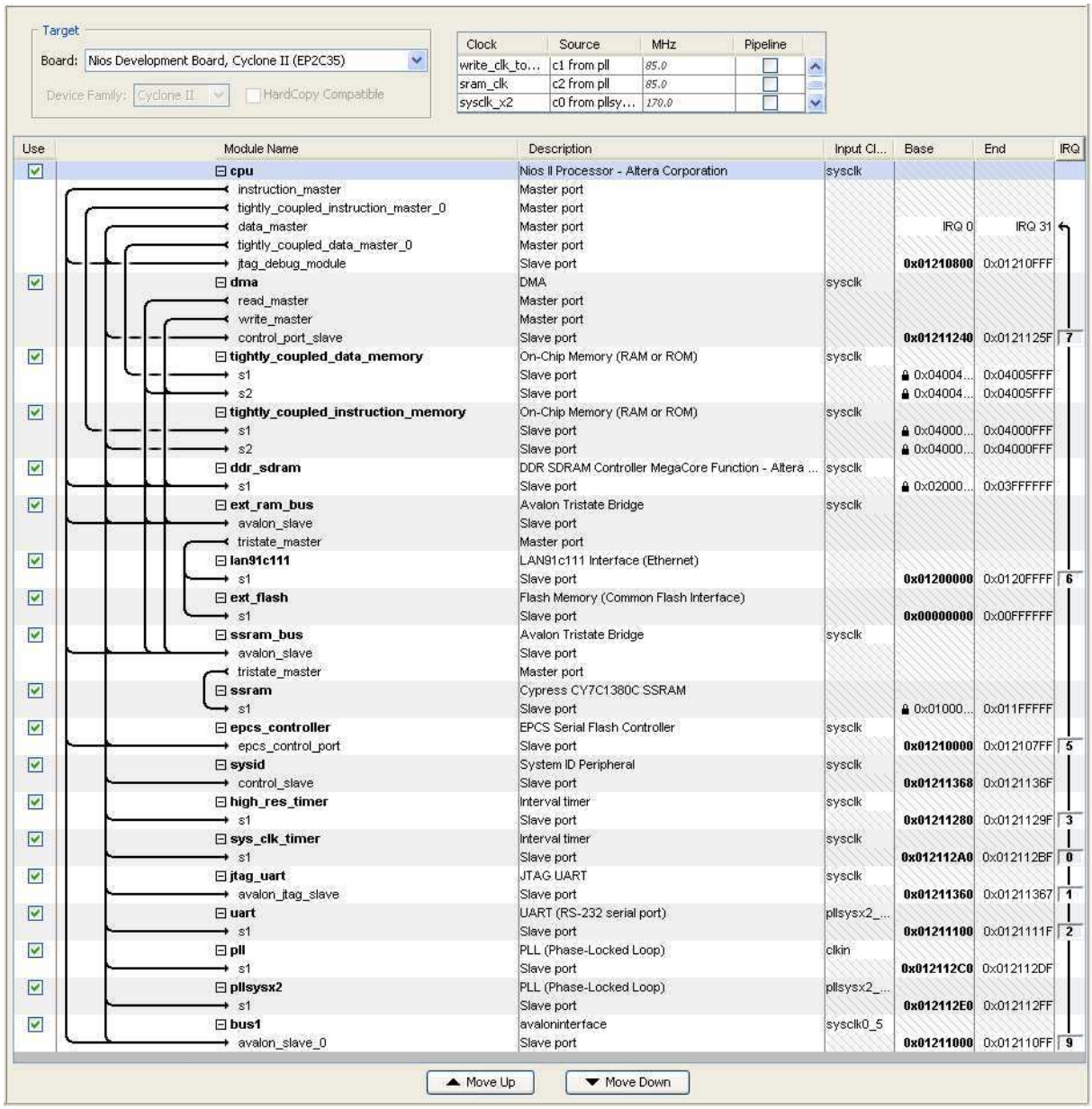


Figure 7.1. Final SOPC Builder system for 2C35 platform

7.2 System interface with external IP cores

Basically the main addition to a predefined system for Nios 2C35 board is the Avalon Interface component, which gives the possibility to the system to later control other cores instantiated outside of the system generated with SOPC Builder, and further expand the design capabilities.
One of the main features of the Avalon Memory-Mapped interface, the proprietary Altera on-chip bus typically used for master/slave connections in Nios II systems, is that there is freedom about which subset of bus signals a component, master or slave, needs to implement, as many signals are only optional, and only a few basic ones are mandatory.
In that particular case, the main idea behind the Avalon Interface component is to be able to create a readable/writable memory map available for peripherals registers. The memory map can be freely organized and used for any other cores instantiated outside of SOPC Builder. That small block also converts the typical Avalon bus to a tristate one, as in some cases, cores will not only be outside of SOPC Builder, but also in other FPGA slave devices. In that case the minimization of I/O pins used, exactly 32 pins, is a very important issue. In that design, it is not so important because the bus is always internal to the FPGA.

Table 7.2 shows a listing of the entire Avalon signals subset used in the Avalon Interface component viewed from the two bus sides.

Avalon-MM Master Bus Side	Dir.	Width	Description	Tristate Bus Side	Dir.
clk	IN	1	The slave component input clock.	ext_clk	OUT
chipselect	IN	1	Used to address a specific slave. When present, the slave ignores all signals unless *chipselect* is asserted. It is always synchronized with a read/write signal.	ext_cs	OUT
read	IN	1	Asserted to indicate an Avalon *read* transfer.	ext_rd	OUT
write	IN	1	Asserted to indicate an Avalon *write* transfer.	ext_oe	OUT
waitrequest	OUT	1	Assserted by the slave in response of an Avalon *read/write* transfer.	ext_wait	IN
irq	OUT	1	Interrupt request signal, asserted by the slave in order to be serviced by the master.	ext_irq	IN
address	IN	6	Represent the address offset from the base address of the Avalon slave.	ext_address	OUT
readdata	OUT	32	The data provided by the slave in response to a *read* transfer	ext_data	INOUT

writedata	IN	32	The data from the Avalon master present in a *write* transfer to the slave.		

Table 7.2. Avalon-MM interface, subset of used signals

After the introduction of the selected Avalon signals subset, some general features can be remarked:

- Master-slaves transfers use a handshaking mechanism instead of fixed wait-states, by using *waitrequest* signal. Slave must assert that signal to finish the transfer, so it can delay the transfer completion as much as necessary. That allows the integration in the system of cores with different speeds.
- With a 6-bit address bus, a maximum of 64 registers of 32-bit width can be addressed and subsequently accessed from Nios II. This is the peripheral memory map reserved to integrate external cores to the system.

In the next chapter, the memory map partitioning strategy is explained, as well as how Nios II controller is reading and writing to registers in cores located outside of the designed Nios II system.

Chapter 8

Hardware Architecture

In the previous chapter it has been explained how the components inside the main NiosII system are connected to the Avalon switch fabric, and it has been introduced how the peripherals outside of the main NiosII system will be subsequently organized in a memory map, in order to allow Nios II to control and configure the slave cores with read/write transactions.

This chapter follows that thread and focuses on the implementation, explaining how the peripherals in the subsystem are organized to achieve that desired behavior, offloading the control an configuration of some of the peripherals integrated outside of the system to the NiosII microcontroller.

Another major issue that needs to be resolved is the clock network design. This chapter explains why plenty of different clocks are needed and how this problem is solved using the FPGA PLL resources and an appropriate clock generation strategy.

8.1 Peripheral subsystem integration with NiosII system

8.1.1 Mixed Avalon/Wishbone system bus

For the components inside the NiosII system, usually the proprietary Avalon Memory Mapped Interface is used, but not all IP cores have a standardized Avalon-MM interface.

Main idea behind any standardized on-chip bus interface is to allow IP cores designers to easily interconnect components in an FPGA and alleviate system integration problems, allowing to reuse properly-designed cores between different projects, improving portability of cores between systems and finally shortening time-to-market. The proliferation of standard bus interfaces is key to the IP cores based designs, as redundant work saved with an IP core can come back as a problems if system integration is not seamless.

A good source of off-the-shelf ready IP cores is Opencores, the largest open source hardware community, and Opencores projects aims to use a common non-propietary bus named Wishbone.

8.1.2 Shared Bus architecture implementation

The main target is to be able to access to the peripheral registers or memory-map although the core uses a Wishbone interface instead of the Avalon-MM interface. Basically Wishbone specification is based in the same principles than Avalon-MM interface specification: master-slave architecture, configurable data and address bus widths, variable subset of signals, handshaking mechanism, etc.

The diagram in Figure 8.1 shows in a graphical way how all the cores are integrated in the system and the different partitions of main system bus. It can be appreciated how all the Wishbone slaves are connected with the Avalon master port in a shared bus topology and with partial address decoding, that means that each slave connected to the bus decodes only the range of addresses that it uses, usually lower bits of address bus. The rest of decoding logic is outside of the slave cores and is used to generate the needed extra *chip enables* for every core

that are used in conjunction with the bus *chipselect signal.* That decoding scheme is very useful in the HDMI generator board, as it allows to integrate cores already designed with different number of registers and variable address bus length without sacrificing any of the memory map available space. For signals from the slaves to the master the shared bus is implemented using different multiplexers controlled by the address decoding logic.

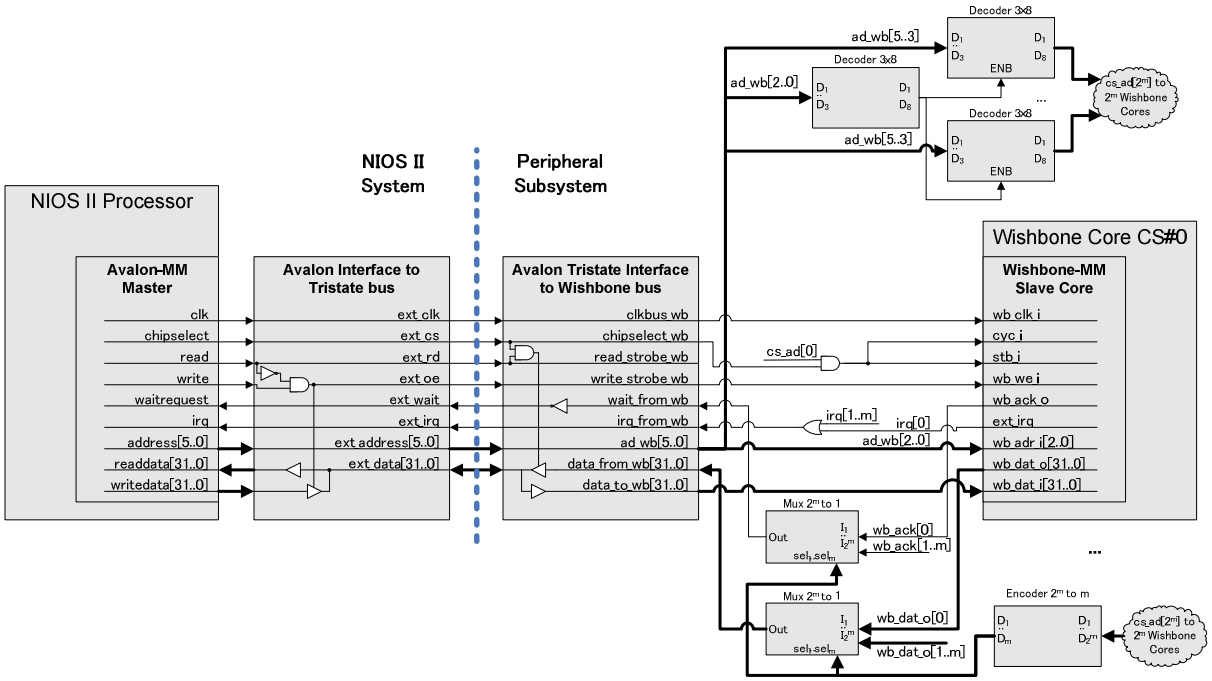


Figure 8.1. System bus architecture. Avalon ↔Avalon Tristate ↔ Wishbone

8.1.3 Examples of bus transactions. Read/Write transfers to slave cores

The following captures uses Altera SignalTap II tool to illustrate the read/write transfers on both Avalon side, in the master port, and Wishbone side, in the slave port. The captures are made to assure the proper implementation of the shared bus topology.

SignalTap II logic analyzer is a tool that allows the system designer to examine the behavior of internal signals without using extra I/O pins and with design running on the target platform. After adding the desired signals to monitor, compiling the design, and downloading it into the target device, data can be acquired using multiple triggers and downloaded to host computer using the JTAG connection for further analysis. Although the necessity to recompile design for any small change makes it less attractive for large projects with high compilation times, SignalTap can be very useful to debug a specific problem, as in this case, where all the bus signals involved are known, and the necessary timeframe to show a complete bus transfer is very short.

Figure 8.2 shows the capture of a read operation in core #0 (address 0x03). The three cycles needed to complete the operation can be clearly appreciated.

0. Master initiates the transfer by asserting *ext_cs, ext_rd* and *ext_address* signals. Core selection signals *wb_cyc_i* and *wb_stb_i* are also correctly asserted, meaning that decoding logic has worked well.

1. Slave core #0 presents valid data from *dout* register in address 0x03 to *wb_dat_o* data bus signal and immediately asserts *wb_ack_o* signal without waiting during extra states. In Avalon side the handshaking signal *ext_wait* is inverted.
2. Master captures data and deasserts the involved signals. Timings are honored.

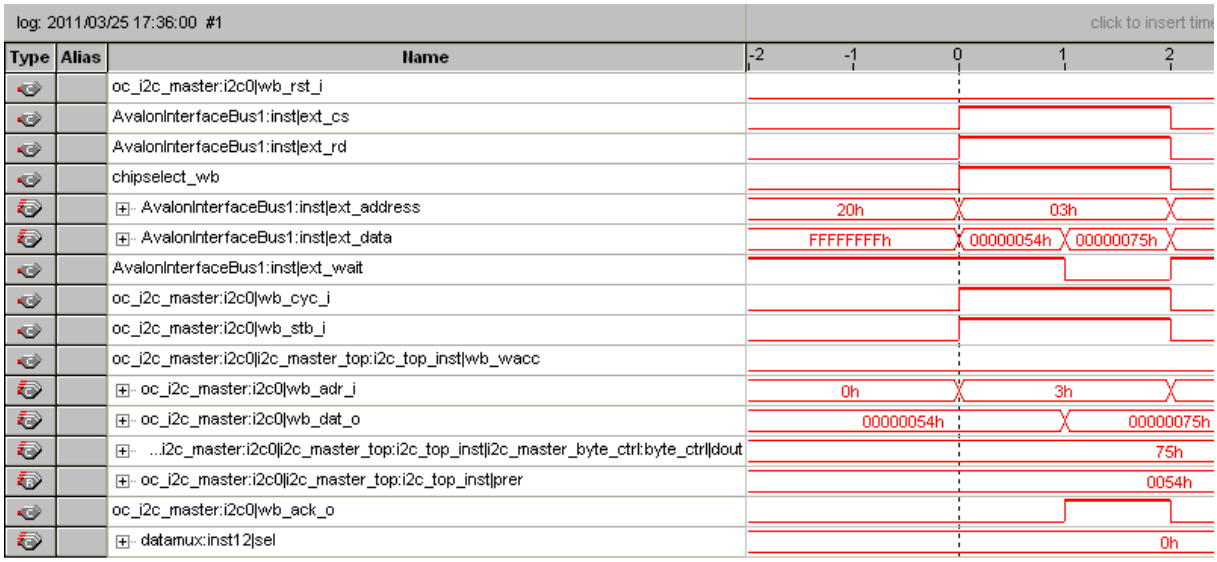


Figure 8.2. SignalTap capture of a read transaction to core #0

For the write operation shown in Figure 8.3, the process is not detailed as it is nearly the same, and it can be appreciated how data from *wb_dat_i* input bus is registered by the slave core and stored in *prer* register at address 0x00.

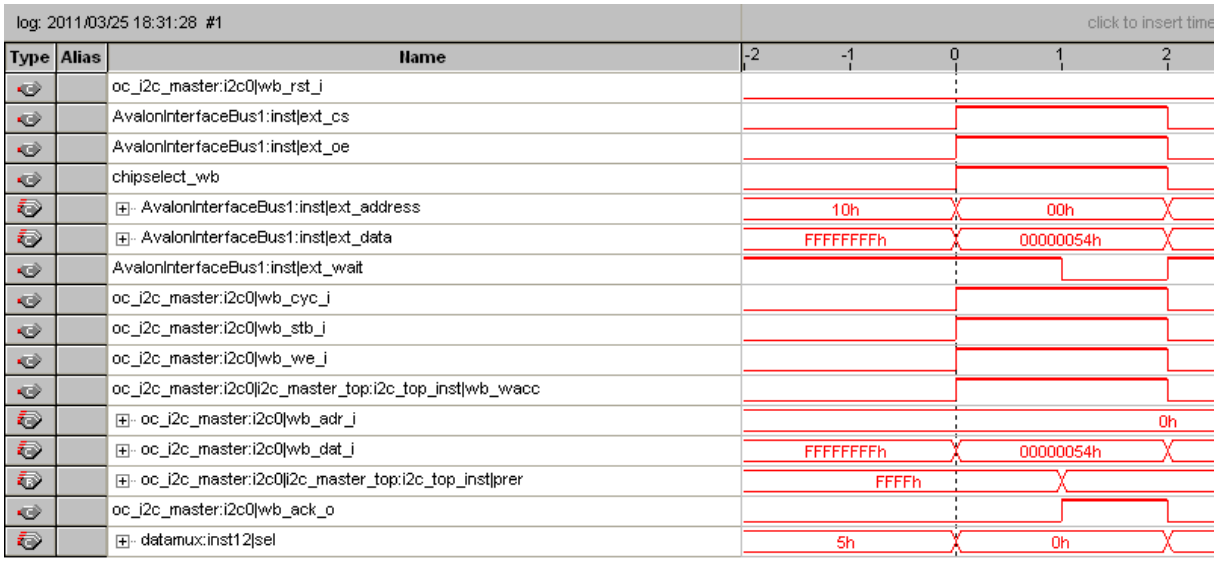


Figure 8.3. SignalTap capture of a write transaction to core #0

8.2 Clock network design

8.2.1 Technology constraints in clock scheme design

Cyclone™ II devices family have up to four enhanced phase-locked loops (PLLs) that provide the possibility to synthesize different frequencies, phase shift and duty cycle programming and other clock management capabilities.
As different blocks in the system need different input clock frequencies not coherent with each other for their correct operation it has been discarded the possibility to synthesize all different frequencies from the same clock source.

PLL blocks in CycloneII devices
The details of the hardware PLL blocks inside the FPGA are outside of the scope of this project, anyway for the correct system integration, a basic introduction of the frequency synthesis capabilities of the Cyclone PLL blocks is needed to understand the decisions taken in this section.

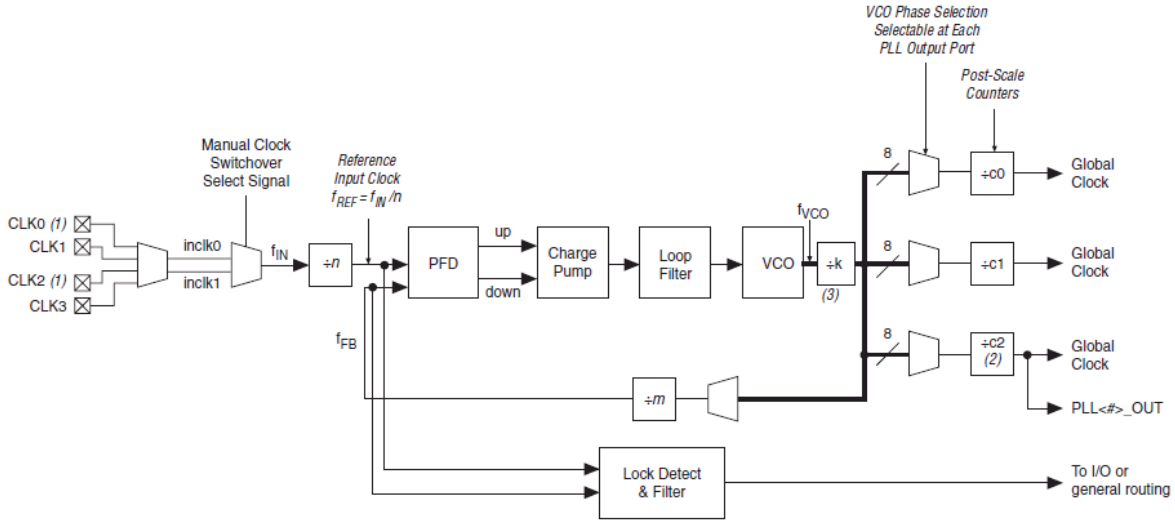


Figure 8.4. Block diagram of a PLL in Cyclone II devices

The CycloneII PLL Block diagram shown in the previous Figure 8.4 is used as a reference to understand PLL operation and capabilities.

- Each PLL block supports up to three clock outputs that route directly to the global clock distribution network. This is a very important detail, because in systems with high clock frequencies the clock skew is a very important issue to take account as it can affect directly to the reliability of the system.
- Each PLL block can be fed by any of four input clock pins in single-ended mode, as is used in the prototype. In conclusion it means in global there are 12 input clock pins that can use the FPGA global clock network.
- Output clock synthesis is achieved using *m/(n x post-scale)* scaling factors. The input clock, f_{IN}, is divided by a pre-scale counter, n, to produce the input reference clock, f_{REF}, to the PFD. This input reference clock, f_{REF}, is then multiplied by the m feedback factor. The control loop drives the VCO frequency to match $f_{IN} \times (m/n)$. Each output port has a unique post-scale counter to divide down the high-frequency VCO. There are three post-scale counters (c0, c1, and c2), which range from 1 to 32.
- The equations for the PLL intermediate frequencies and the synthesizable frequency outputs, depending on the PLL scaling factors, are shown in (8.1.)

$$f_{REF} = \frac{f_{IN}}{n}$$
$$f_{VCO} = f_{REF} \times m = f_{IN} \frac{m}{n} \qquad (8.1.)$$
$$f_{Cx} = \frac{f_{VCO}}{Cx} = f_{IN} \frac{m}{n \times Cx}$$

In conclusion it can be seen that there are potentially 12 clocks that can be routed to global clock network; however there is not complete freedom to select *n, m and Cx* scaling factors, thus limiting the generation of different clocks to a short range of synthesizable clocks.

8.2.2 Input clock requirements for different blocks

System

The clock used in the NiosII must be synchronous with the clock used to drive the Avalon slave components in the system and also with the cores external to the system that are hanging on the Wishbone bus. As the system clock has to be used for the control plane of the slave cores, it creates a potential problem as a different clock domain must be used in the audio and video generation blocks for the data plane. That cross-clock domain issue is a risk factor for the correct behavior of the system and it cannot be treated carelessly, as metastability problems can make the design very unreliable. Some of the techniques to synchronize properly the cross-clock signals between the control and data planes, must be used when designing that blocks.

Video generation (data plane)

The video block is required to generate appropriate timings for all the video formats defined in the EIA/CEA-861D specification, and that the ADV7511 transmitter is able to detect. CEA-861D specification will be explained in more detail in the corresponding video generation section, but in a short introduction it defines a variety of video timings that are usually defined for a number of parameters including the pixel frequency or pixel rate. An example of typical video timings can be found in the Table 8.1, for example, Video Identification Code (VIC) #31:1920x1080p @50Hz, typically known in the shorthand abbreviate version of 1080p, and very popular as the main FHD video format supported in FHD displays in Europe. The other two example correspond to other popular video timings in mass-media video sources and displays, like 720p or 576p

VIC	Hactive	Vactive	I/P	Htotal	Hblank	Vtotal	Vblank	H Freq (kHz)	V Freq (Hz)	Pixel Freq (MHz)
31	1920	1080	Prog	2640	720	1125	45	56.250	50.000	**148.500**
19	1280	720	Prog	1980	700	750	30	37.500	50.000	**74.250**
17,18	720	576	Prog	864	144	625	49	31.250	50.000	**27.000**

Table 8.1. Example of pixel clocks required for some popular video timings.

As the main objective of the video generation block is to feed the ADV7511 with all the available video formats in CEA-861D specification, the input pixel clock in that block must be changeable between different input clocks present inside the FPGA.

Audio generation (data plane)

ADV7511 is capable of receiving digital audio for transmission over HDMI in a lot of different formats. Specifically I^2S format has been selected to generate internally to the FPGA some digital audio tones

I^2S protocol operation will be further explained in the correspondent chapter about audio generation block, but as a short introduction, it typically consists in three clocks and one data lines. Data is running synchronized with the bit clock (BCK or SCK), that is a multiple of the data sampling rate. This multiple depends of the number of bits per sample and number of stereo channels. For example in an application running at CD audio sampling frequency of $Fs = 44.1KHz$, with 32 bits and two channels, the bit clock will run at $32 \times 2 \times 44.1KHz = 2.8224MHz$.

The audio master clock reference that is a multiple of bit clock, and runs at a higher rate, is typically used for advanced audio processing with digital interpolation filters, and is not mandatory in all I^2S systems, for example simple DACs do not use it, but ADV7511 needs this input reference clock, that can be selected as a multiple of Fs $128 \times N \times Fs$ with $N = 1,2,3,4$

Following the previous example with CD Audio sampling rates, the input reference clock should run for example at $128 \times N \times 44.1KHz \left| (N = 4) = 22.579MHz \right.$

ADV7511 support a big amount of sampling rates up to 192KHz including $Fs = 44.1KHz$, so the audio block also must be able to switch between different input reference clocks in order to generate digital audio at different sample rate frequencies.

DDR & SSRAM

These external memories in the 2C35 development kit need each one their respective clocks, using in this way some of the PLL resources. For DDR SDRAM, the same IP configuration tool used from SOPC Builder generates automatically the PLL with a buffered clock, and sets all the phase offsets of the relevant clocks, so this part is only introduced to remark that some resources are already used. The same can be applied to the SSRAM clock, which in a normal case should be configured with a precise phase relationship with the system clock, after a timing analysis, but in the prototype is already configured as the design is based on one of the development kit examples.

8.2.3 PLL resources usage & configuration. Clock distribution network scheme

After analyzing the technology constraints and the clock requirements of the main system blocks, is easy to notice why the option to generate all the different clocks using only one input clock source together with the PLL resources has been discarded.

The 50MHz system clock is going to feed two of the four PLL in the Cyclone II device, and the different coherent internal clocks synthesized will be used to drive different blocks of the system, like the NiosII processor, DDR, SSRAM, and the control plane of the slave IP cores.
The 27MHz oscillator physically located in the daughter card and transmitted through the proto connectors is used as the input clock source to generate the different video clocks used as the pixel clock source for the video generation block data plane.
The 16.384MHz oscillator physically located in the daughter card and transmitted through the proto connectors is used as the input clock source to generate the audio master reference clock used in the audio generation block data plane.

The following block diagram in Figure 8.5, shows the configuration for each one of the four PLLs present in the FPGA, with their correspondent scaling factors.

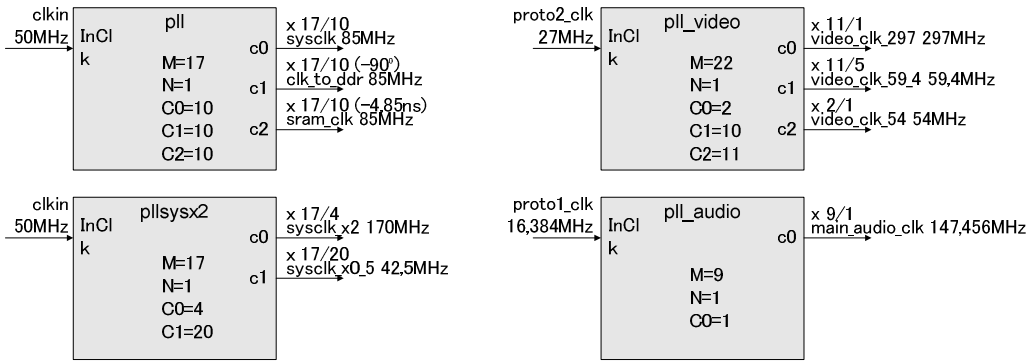


Figure 8.5. Definitive PLL input and output clocks configuration.

Nearly all the resources for clock synthesis have been used and although generated clocks are still short of the objectives presented before, some of the other needed frequencies can be generated using common synchronous dividers by power of two or even integers, keeping all the clocks synchronous and with a 50% duty cycle.

For the video generation block, only three pixel clock frequencies, shown in Table 8.2, remain impossible to synthesize with this scheme, but it is impossible to avoid this trade-off, and the video timings that will remain impossible to generate are only a few ones, even in some cases deprecated special timings.

Clocks Synchronous to 108 MHz			**Clocks Synchronous to 297 MHz**			**Other Legacy clocks in CEA-861E**		
Pixel Clock	**PLL Out**	**Ratio**	**Pixel Clock**	**PLL Out**	**Ratio**	**Pixel Clock**	**PLL Out**	**Ratio**
27 MHz	pll_c2/ 2	22/(1*11)*1 /2	**59,4 MHz**	pll_c1	22/(1*10)	**25,175 MHz**	Not Synth	538/577 (*)
54 MHz	pll_c2	22/(1*11)	**74,25 MHz**	pll_c0/ 4	22/(1*10)*1 /4	**72 MHz**	Not Synth	8/3 (*)
108 MHz	Not Synth	4/1 (*)	**148,5 MHz**	pll_c0/ 2	22/(1*10)*1 /2			
			297 MHz	pll_c0	22/(1*2)			

Table 8.2. Synthesizable frequencies for video pixel clock

For the audio generation block a different strategy is used as one master clock is generated in order to synthesize all the other frequencies with simple dividers by even integer numbers, very easy to generate in VHDL using toggle flip-flops.
In the Table 8.3 there is the summary of the generated synchronous clocks accepted by the ADV7511, that can be achieved from a 147,456MHz master reference clock, synthesized with the PLL from the 16.384MHz input clock.

Sampling Freq.	**Bit Clock Frequency (BCK)**	**System Clock Frequency (MCK)**				
	64 F_S	**128 F_S**	**192 F_S**	**256 F_S**	**384 F_S**	**512 F_S**

(F_S)	Freq. (MHz)	Div	Freq. (MHz)	Div	Freq. (MHz)	Div	Freq. (MHz)	Div	Freq. (MHz)	Div	Freq. (MHz)	Div
8 kHz	0,512	1/288	1,024	1/144	1,536	1/96	2,048	1/72	3,072	1/48	4,096	1/36
16 kHz	1,024	1/144	2,048	1/72	3,072	1/48	4,096	1/36	6,144	1/24	8,192	1/18
32 kHz	2,048	1/72	4,096	1/36	6,144	1/24	8,192	1/18	12,288	1/12	16,384	1/9
44,1 kHz	2,8224	4/209	5,6448	8/209	8,4672	52/923	11,2896	49/640	16,9344	41/357	22,5792	49/320
48 kHz	3,072	1/48	6,144	1/24	9,216	1/16	12,288	1/12	18,432	1/8	24,576	1/6
88,2 kHz	5,6448	8/209	11,2896	49/640	16,9344	41/357	22,5792	49/320	33,8688	147/640	45,1584	49/160
96 kHz	6,144	1/24	12,288	1/12	18,432	1/8	24,576	1/6	36,864	1/4	49,152	1/3
192 kHz	12,288	1/12	24,576	1/6	36,864	1/4						

Table 8.3. Synthesizable frequencies for audio master reference clock

Two of the sampling rates accepted by the ADV7511, typical Audio CD sampling rates of 44,1KHz and 88.2KHz, have been dropped as it is not possible to synthesize the needed frequencies without another extra PLL, that it is not available in the current design. Anyway the typical audio sampling rates sent over HDMI by common HDMI sources are the DVD-Audio sampling rates of 96KHz and 192KHz, so although there is a trade-off, the main objective to be able to have highly configurable audio clocks is achieved.

In summary, the extra input clocks and a smart PLL configuration in combination with some useful configurable clock divider cores, can achieve a big part of the clock frequencies for video and audio generation needed for this design, with only some little trade-offs.

Chapter 9

Cores development & system integration

In the previous chapter it has been explained how external cores can be controlled by the NiosII system although integrated outside of the SOPC Builder generated system.
It means that the topic is focused on how to integrate the cores, make appropriate connections according the shared bus strategy and interact with the controller, but without the chapter objective is based on how to make the system integration but does not detail the needed cores in the HDMI generator board.

This chapter introduces all the cores, shown in Figure 9.1 block diagram, needed to achieve a functional prototype with all the desired features, as all these cores are needed to interact in one way or another with the ADV7511 HDMI transmitter by feeding it with audio and video signals, as well as providing some control mechanisms. The explanation focuses on cores entirely developed from scratch, but also briefly explains all the cores integrated in the system from public core repositories like OpenCores, as design reuse and easy integration are always some of the strongest points of this kind of FPGA systems based on NiosII.

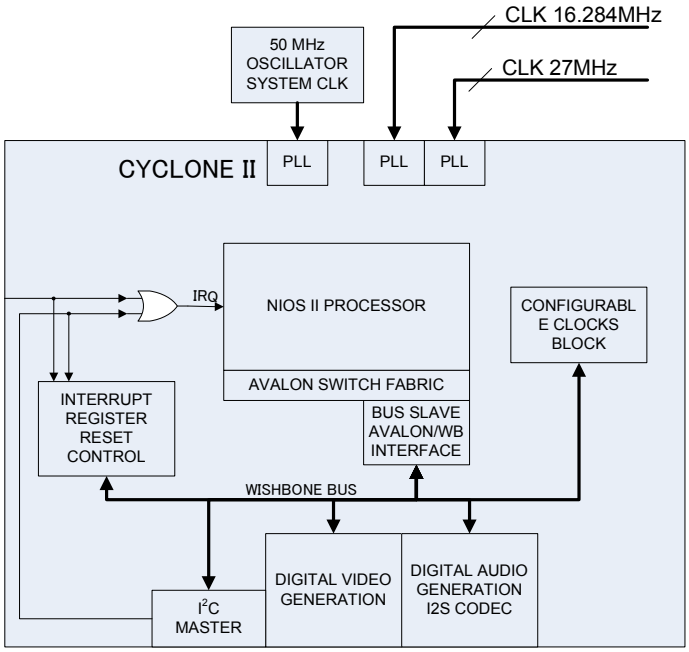


Figure 9.1. Block diagram of all cores implemented and connected to NiosII

Finally, the registers map of all the cores explained in this chapter and integrated in the HDMI generator board, is included in *Annex F: HDMI Generator Board Cores Reference. Registers Map & I/O Signals.* Although it is not strictly necessary to understand the core development, it is a useful support material.

9.1 I^2C Master

The I^2C Master core is the main control block integrated in the system, as the ADV7511 registers must be programmed through the SDA and SCL pins using the I^2C protocol. For this purpose instead of creating from scratch a I^2C master core, a component from Opencores library has been used *[14]*.

Figure 9.2 shows how the I²C master core Wishbone interface is connected straightforward to the Wishbone bus, as the system has already been prepared to make integration as easy as possible. With this scheme NiosII will use direct memory access to act as the core controller.

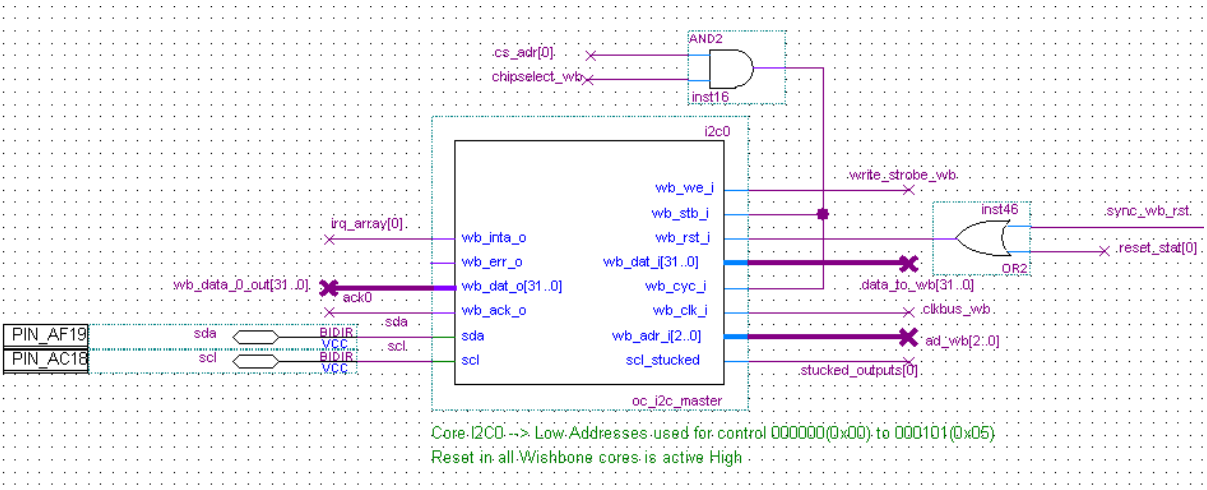


Figure 9.2. I²C core integration with the system

In order to make minimal modifications, as the original core used only three address bits, the decoding logic has been adapted to enable the core every time the NiosII makes a read/write transaction to addresses from 0x00 to 0x05.

As the integration from hardware point of view is very simple and all the interaction is made with the NiosII, the registers programming sequence to operate with the core will be explained from software point of view in the corresponding software chapter.
The registers memory map is included as a reference in *Annex F.1: I2C Master Core.*

9.2 Video generation

9.2.1 Introduction

The ADV7511 HDMI transmitter used in the design has a video capture block with an uncompressed digital video interface that accepts and recognizes all the **video timings** defined in the CEA-861-E specification. Generally, the ADV7511 registers can be programmed to accept all the other different video format options that are not implicit with the received signals present in the digital video interface, but needed to determine the video format, as bits per color component (**color component depth**), input **color space** and **subsampling** scheme from RGB 4:4:4 or YcbCr 4:4:4 to YcbCr 4:2:2 and different **picture aspect ratios.**

ADV7511 acts for one side as a video sink, receiving video, in a video format defined in the specification, from a CEA-861-E compliant source over a raw digital video interface. For the other side it really acts as a video source, sending video in the same video format received but over another physical interface, the HDMI link.

This section is focused on the development of the video generation core and its general operation. As the core is a CEA-861-E compliant video source that feeds ADV7511 video capture interface, in *Annex C.2: Digital Video concepts,* a deep explanation of the CEA-861-E

specification as well as the fundamentals of raw digital video can be found. These basic concepts can be very useful to complement the core development explanation.

9.2.2 Core internal architecture & hierarchy definition

It is concluded that a video timing can be generated by following strictly the specification for active/blanking pixels/lines for the desired VIC and clocking the video data with the appropriate pixel clock frequency associated to the VIC.

As the core has to be entirely configurable at execution time it means some registers are needed to store all these information that defines a video timing. So the core is built around a wishbone interface that acts as the top hierarchical block. All the registers readable and writeable using the wishbone bus must be completely mirrored in the pixel clock domain, which works with a clock different than the system clock used by the Wishbone interface. There are several techniques to avoid metastability problems derived from cross-clock domain signals, but it will be explained later why a dual clocked FIFO has been used to avoid this issue. Finally the video timings block generates the video timings waveforms using the information configured in the registers, and the color bar generator block uses these signals to clock data from internal memory and create very simple patterns.

The video timings block and the color bar generator block have been manually instantiated in the top hierarchical block, the video generator wishbone interface, as well as the dual-clock FIFO megafunction from Altera. As the connections between blocks have also been configured by assignments instead of by wires in the schematic view of Quartus II, the block diagram in Figure 9.3 has been created to appreciate the connections between the blocks that compose the core in a graphical way.

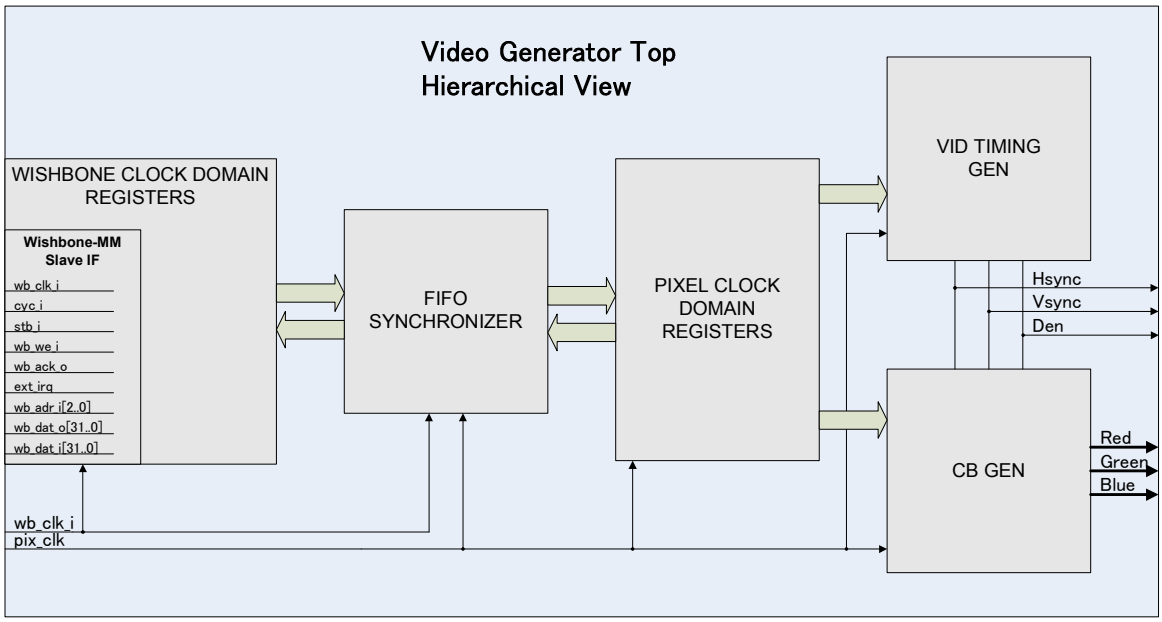


Figure 9.3. Block diagram of core internal hierarchical view

9.2.3 Core theory of operation

9.2.3.1 Wishbone slave interface implementation

As it has been explained before, the core has a wishbone interface to modify all the configuration information used to generate the timing waveforms stored in registers.
The registers memory map is included as a reference in Annex *F.2: Video Generation Core.*

The wishbone slave block has been designed in the simplest way to fulfill all the timings in wishbone bus specification.

First of all an internal wishbone write access signal has been created as a combination of all three of the chipselect inputs, *wb_cyc_i* and *wb_stb_i* and *wb_we_i*. That's the simplest way to implement it, as the specification remarks that all three of these signals need to be in active high state to determine that the write transaction is addressed to this core. The external decoding logic takes care of activating the chipselect inputs when transaction is really addressed to the core.

When the master initiates the transaction, valid data is present in the *wb_dat_i* bus. The implementation assures that data is stored in the internal register associated with the address present in the bus *wb_adr_i,* just one cycle after the internal wishbone write access signal goes high. As it has been explained in previous chapters that is the expected behaviour for wishbone bus.

For the read transactions it has been decided to still make a simplest implementation to save any useless waste of resources. All the *wb_dat_o* signals from the cores in the system go to a bus multiplexer controlled by logic external to the cores, so the core data bus outputs can not interfere between each other, so it has been decided that the core will always present the data associated with the wb_adr_i input to the data bus output, just one cycle after the address is present in the bus. That means that there is no direct control to detect if the transaction is a read one or addressed to the core, and data will be presented also when the transactions are really addressed to other cores, causing some extra transitions in this signal that are not going to affect the complete system behaviour. In fact the master really knows when a read transaction has been started, and data in the bus is going to be correct when the master takes it one cycle after the transaction start.

Finally the handshaking signal *wb_ack_o* is also modified inside a process statement clocked at *wb_clk_i* input clock to honor wishbone timings and rules. Ths signal is always acknowledged one cycle after *wb_cyc_i* and *wb_stb_i* goes simultaneously high and only lasts one cycle in high level. The wishbone capability to use this signal to stall momentarily the transaction initiated by the master is not used in this implementation.

The captures in the *Chapter 8.1.3: Examples of bus transactions. Read/Write transfers to slave cores*, showing bus transactions, can be used as a reference to understand this implementation.

9.2.3.2 FIFO synchronizer

As it has been repeated in the section 8.2 about clock network design, the use of two different asynchronous clock domains can not be avoided, as system & data clocks have variable phase and time relationships. It is also unavoidable that some data needs to be shared between them, so it has to be made with great caution. In a nutshell, if the data input of the storage element changes too close to a clock edge, the element may go into metastable state and the output cannot be reliably used. That situation can potentially happen in cross-clock domain situations if the signals are not asserted long enough and registered, because signals appear asynchronously in the incoming clock boundary. To learn more about metastability and his harmful effects for the reliability of electronic circuits some references are provided *[35][36]][37]*.

There are several techniques to avoid these issues, basically based in the concept of "buy time" in order to settle the metastable-prone signal to a value; usually that process is simply called synchronization.
The typical solution when synchronizing a signal from a slower to a higher frequency clock domain is usually based on adding successive stages of cascaded synchronizing flip-flops clocked at the higher frequency. This approach cannot guarantee that metastability is avoided, but reduce the probability to tolerable levels, usually with a double FF synchronizer it is enough, although this two stages also add some latency when following input changes.
Another solution that is usually made when a data flow is needed is the use of a dual-clock FIFO with different read/write clocks.
In the design it has been decided to use the dual-clock FIFO because it has several advantages in front of the double FF synchronizer alternative.

- FIFO already integrates by design the necessary logic to manage properly the clock domain crossing problems *[38]*
- The double FF synchronizer cannot be used properly to avoid metastability if the signals to synchronize go from a higher to slower clock domain. Pixel clock frequency is variable and potentially, in the worst possible scenario, it can be configured to 27MHz, compared to the 42.5MHz frequency of the system clock, so that probable situation advise against the use of double stage FF synchronizers.

Figure 9.4 shows the FIFO block used in the design, a parameterizable dual-clock FIFO (DCIFIFO) Altera megafunction. The *wrreq*, *wrfull* and *wrempty* signals are synchronized to write clock *wrclk*, and the same happens for the signals in the read side and read clock.
The FIFO is loaded from the packed registers of the wishbone clock domain side after any change in their contents. This behavior is achieved by using the same internal wishbone write access signal to start the FIFO write sequence in the next clock cycle, after the contents of the register have been updated.
Figure 9.4 also shows the state diagram of the operation to read from FIFO in the pixel clock domain. The block is initially in the *idle state*. When data is loaded from wishbone clock domain and is ready to be read from the pixel clock domain the *rdempty* signal goes from high to low and that starts the read transaction. The block changes to *pre_read_fifo state* and the *rdreq* signal is asserted during one cycle to make the FIFO present the data in its output. The block changes directly to the *end_read_fifo* state where *rdreq* signal is deasserted, and data is registered again in the pixel clock domain. During these moments, where register values are changed, the cores running at the pixel clock frequency remain in a reset state, as it is preferred to start the generation of new timings when all the information is already correctly updated, as the cores use the registers to generate the correct waveforms for the digital video outputs and some strange artifacts could appear in the video, so in the *end_read_fifo* state the *core_disable* signal is finally deasserted allowing cores in the pixel clock domain to run using the new configuration information.

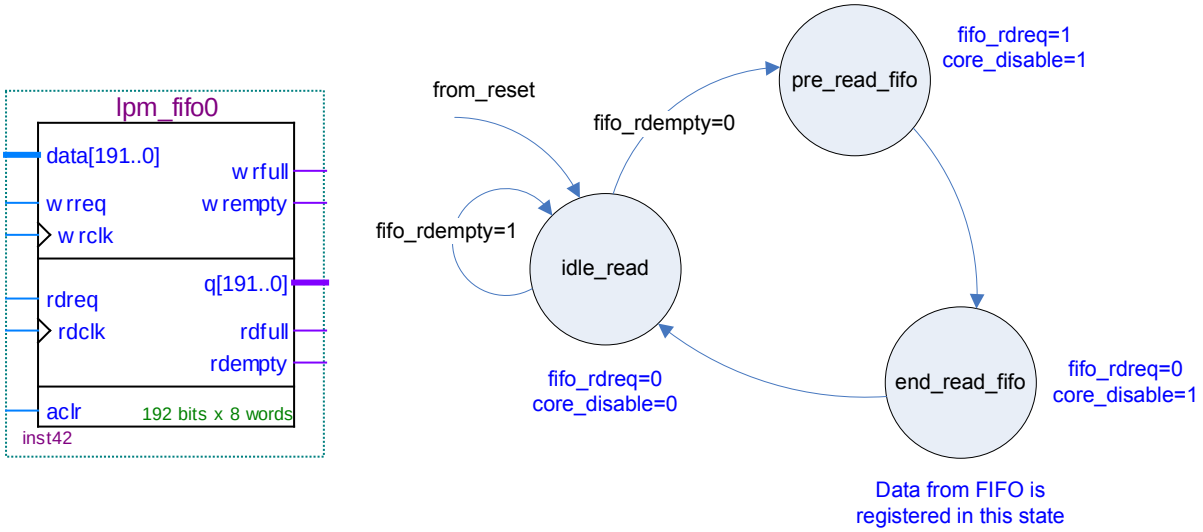


Figure 9.4. Dual-clock FIFO symbol and state diagram representing FIFO read block from pixel clock domain

9.2.3.3 Video timings generation

This is the main block of the video generation core as it is responsible to generate Hsync, Vsync and Data Enable signals. The block is able to generate waveforms for both progressive and interlaced video and with any sync polarity, but in order to simplify the explanation of the implementation, only the progressive video generation with positive sync pulses, corresponding to Figure 2 of CEA-861-E specification, will be deeply defined as interlaced video is a bit more complex and it will be explained only by remarking the main differences. All the informational and support material in *Annex C.2: Digital Video concepts*, can also be used to illustrate the operation of this block.

First of all, the core starts in a known state after reset or any update in the registers, as the *core_disable* signal coming from the FIFO synchronizer block marks that a new configuration information has been loaded from the FIFO.

The conditions checked for state transitions are the horizontal counter, *hcnt*, which counts pixels in a line, and the vertical counter, *vcnt*, which counts lines in a frame. In the Figure 9.5 there is a timing diagram that can be used to understand the different state transitions depending on the horizontal and vertical counter values. The timing diagram is represented in two axis to show all the states involved during a progressive video frame. As the process is clocked at pixel clock frequency, the *hcnt* internal register is updated after every pixel clock cycle, and *vcnt* internal register is updated when there is a edge in the horizontal sync signal.

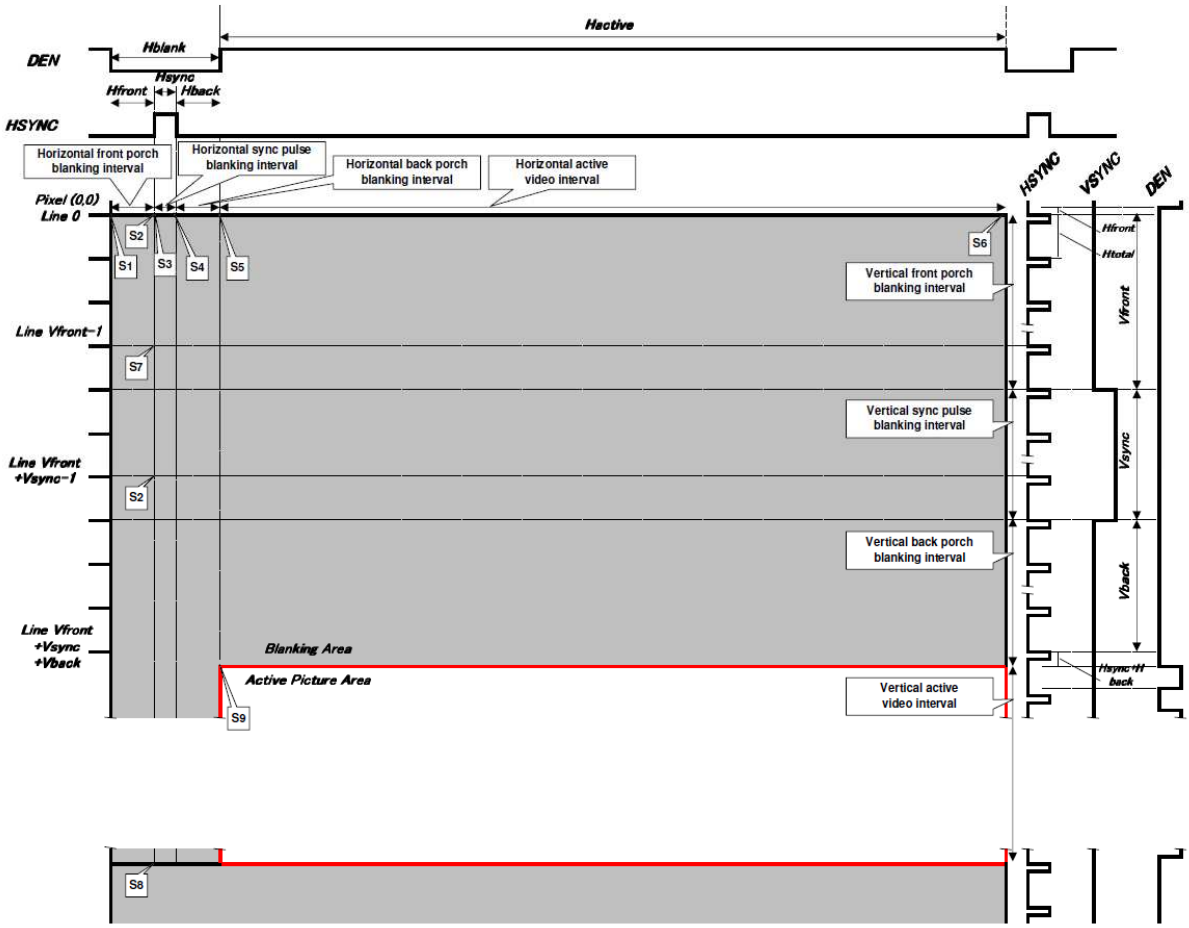


Figure 9.5. State transitions involved in generation of a progressive video frame represented in a time scale with the core outputs evolution

The 9 different state transitions represent the different intervals in the 2D video frame, front porch blanking, sync pulse blanking, back porch blanking and the active video period.
It has to be noted that blanking periods are the combination of the horizontal and vertical blanking intervals, as well as the active video period is the overlapping of the active pixels interval in a line with the active lines in a frame. The Figure 9.6 shows the state transitions diagram in a more typical way, representing all the conditions checked to change between states as well as the value of the outputs in each state. The combination of these two diagrams is made to help the reader in the temporal evolution of the states during the generation of the timings that represent a video frame.

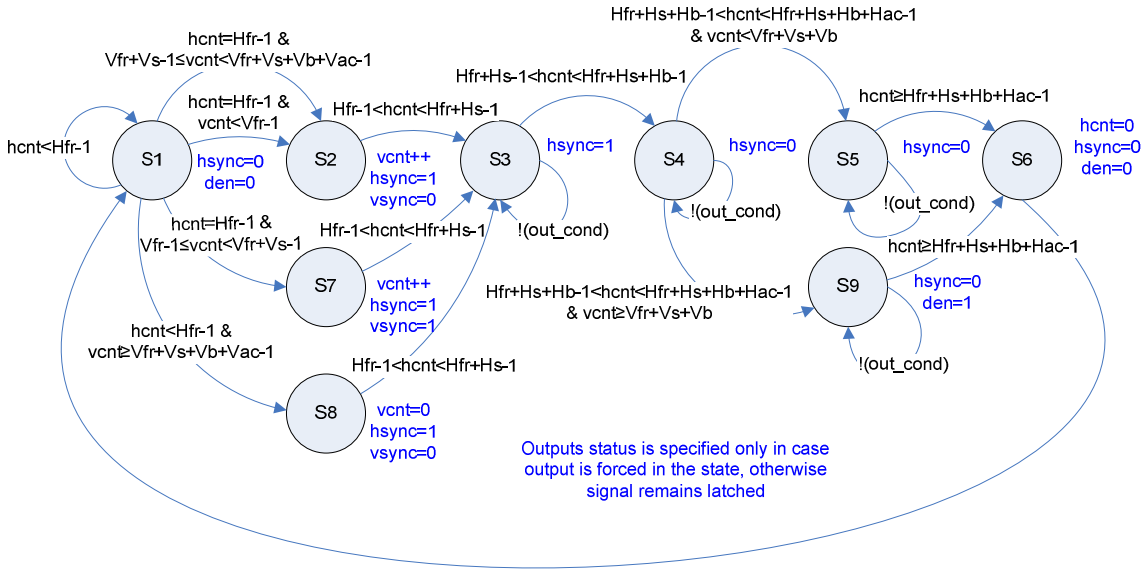


Figure 9.6. State transitions diagram for the video timings generation block

Graphical representations are self-explanatory and give a big help to understand the development explanation that follows. After reset or registers change, where the *hcnt* and *vcnt* counters are cleared, the cores always goes to state *S1*, this state corresponds to the front porch of blanking time before the horizontal sync pulse in a video line, and the core remains there as long as *hcnt* is less than the Hfront-1 value in pixels configured in the appropriate register for front porch blanking. In this state *hsync* and *den* signals are forced low, but *vsync* signal remains latched, keeping the old value, as it is possible that this line is inside the vertical sync pulse, and is not possible to assure the value without also checking the state of *vcnt* counter. It is also important to remark that the comparison of the *hcnt* counter is always going to be made with the value in the registers minus one, because the assignments in a process do not have an immediate effect, and to count exactly the number of pixels in every interval of the video line, the comparison has to be made in advance.

The core goes to next state and leaves the front porch blanking, when *hcnt* is equal to Hfront-1 pixels. In ths state, the *hsync* signal goes high to start the horizontal sync pulse. Depending on the value of *vcnt,* the vertical sync pulse *vsync* will also go high synchronously with *hsync.* In this state, the *vcnt* counter is also increased, or rolled cyclically to 0 if the state corresponds to the last line of the frame. The core only remains in this state during one cycle.

The core goes to state S3, corresponding to the horizontal sync pulse blanking time, when the condition Hfr-1<*hcnt*<Hfr+Hs-1 is satisfied, and in this state the *hsync* output remains high. When the *hcnt* exceeds Hfr+Hs-1 the core goes to state S4, corresponding to the last interval of blanking, the back porch blanking period. In this state the *hsync* output goes low. Finally when the *hcnt* exceeds Hfr+Hs+Hb-1 pixels, the core leaves the blanking interval, marking the start of active video if the line is also active. To check if the line is an active line the core checks if condition *vcnt* ≥Vfr+Vs+Vb is fulfilled. In that case the core goes to state S9, that marks the start of the active video in the video frame, and signal *den* goes high, otherwise *den* signal keeps the last value.

Finally after the *hcnt* pixel counter exceeds the total pixels for the line, Hfr+Hs+Hb+Hac-1, the *den* and *hsync* signal are forced low and the pixel counter is rolled back to 0 to start the generation of a new frame.

As a conclusion it is important to remark than the behavior of the horizontal blanking interval and the vertical blanking interval are very similar, counting pixels to define the different intervals in the lines for the horizontal blanking interval, and counting lines to define the different intervals in the frame for the vertical blanking interval.

The basic concepts used to generate the progressive video timings are also used to generate interlaced ones, although in that case there is some additional complexity to the process. Basically an internal signal *interlaced_frame* is used to toggle between odd and even frames at the end of every frame, as the behavior is very different in the two cases. In case of odd fields, it is exactly the same than a progresive video timing frame, but with half of the active lines, and synced *hsync* and *vsync* pulses. In the even frames there is an extra blanking line that is added half in the vertical front porch and half in the vertical back porch intervals. In this frames the *vsync* pulse is not synced with *hsync* pulse, as it starts exactly a half-line (Htotal/2) after the *hsync* pulse. All these additional issues generate more states, but the idea to generate timings by comparing pixel and line counters with the different active/blanking intervals is exactly the same.

9.2.3.4 Video patterns generation

As it has been explained before in the video timings generation block, data in video outputs is only valid and sampled by the video sink during the active video interval when *den* signal goes high, so this block is the responsible to generate some simple test patterns of low complexity and small memory requirements by clocking data in video outputs during the active video interval.
As explained in *Annex C.2: Digital Video concepts*, a video format is defined not only by the video timing, but also by how the pixels are encoded in the digital data lines, so in this matter some decisions have been made to reduce the complexity of this block.
About **color space** and **data subsampling**, there is no extra benefit to be able to send data between the FPGA and the ADV7511 using different color spaces, as the transmitter also has a powerful color space converter, that is able to generate different pixel encodings over HDMI from the same source pixel encoding, so this is the main reason to constrain the pixel encoding options to RGB 4:4:4 in order to reduce complexity for a feature that is not going to be used. Another important advantage is that in the HDMI 1.4 specification the only pixel encoding required as mandatory for all sources and sinks is RGB 4:4:4, which means that most of the times the color space converter in the transmitter will not be required, reducing overall system complexity.
Other important parts of the specification for this block, which need to be constrained, are the **color depth** and the **quantization range**.
The ADV7511 accepts from 8 to 12 bits/component, but data mapping really can be done by "software" configuring the appropriate internal registers. So in order to be prepared for the worst case scenario of complexity, it has been decided to use all 36 data pins to generate the patterns. Although it seems that 12-bit color depth is useless in the very simple patterns generated, that option is really testing some extra features of ADV7511, because for deep color formats from 30-bit mode to 36-bit mode, the pixel packing in the HDMI link is different than in common 24-bit mode. Basically for a color depth of 24 bits/pixel, TMDS clock runs at the same rate than pixel clock, but for deeper color depths, the TMDS clock runs faster than the source pixel clock, providing the extra bandwidth for the additional bits.
In the Figure 9.7 there is a graphic representation of the final condition for a 12-bit color depth and RGB 4:4:4 pixel encoding.

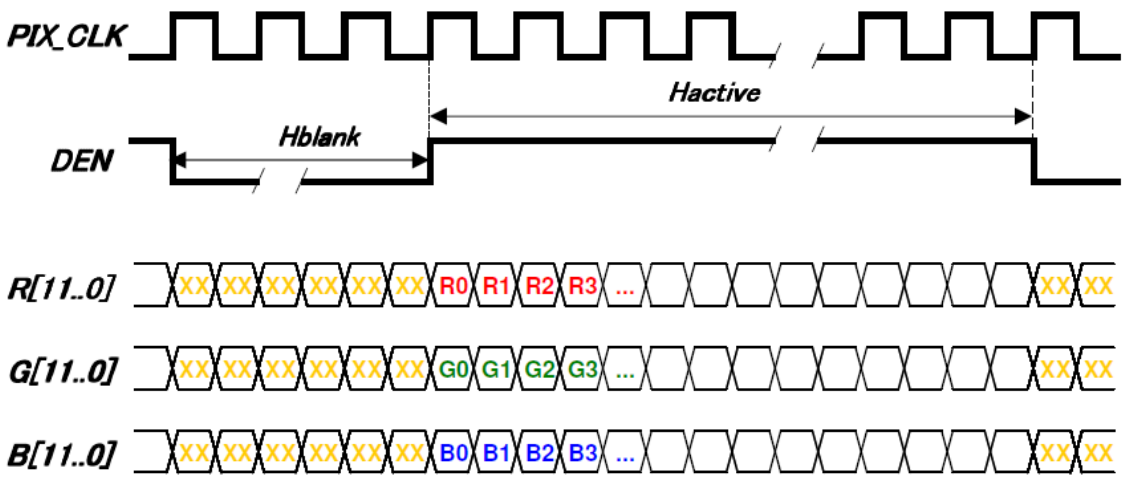


Figure 9.7. Default Pixel Encoding: RGB 4:4:4, 12bits/component

Finally, the limited video quantization range will be used, with some coded values in the extremes excluded, because the HDMI 1.4 specification *[20]* remarks that Limited Range shall be used for all video formats defined in the CEA-861-E specification *[19]*, with the exception of 640x480 format, that uses full range.

In practical effects that means than black and white levels in the extremes have to take different values than 0 and 2^n-1 bits. In the Table 9.1, extracted from HDMI 1.4 specification, the recommended ranges are specified.

Color Component	Component Bit Depth	for Full range		for Limited range		
		Black level	Nominal Peak (White level)	Black level	Nominal Peak (White level)	Valid Range
R / G / B	8	0	255	16	235	1 to 254
R / G / B	10	0	1023	64	940	4 to 1019
R / G / B	12	0	4095	256	3760	16 to 4079
R / G / B	16	0	65535	4096	60160	256 to 65279

Table 9.1. Video Color Component Range vs Color Depth for RGB components

The block is able to generate some different patterns, but basically the explanation will be concentrated in the most typical one, the horizontal color bar with 100% color intensity as the others are generated using basically the same techniques. In order from left to right, the colors in the bars are gray, yellow, cyan, green, magenta, red, blue and black.

For the color bar pattern, the block only needs to know about the horizontal position as all the active lines in the frame are exactly the same. In brief words, just after active video period starts, when *den* signal goes high, an internal bar pixel counter signal *bar_pix_cnt* starts to be increased every cycle, until the bar pixel counter matches the expected width of the bar. At this point another internal signal used to count the already generated columns, *bar_cnt*, is updated. In the pre-initialized lookup table that is shown in Figure 9.8 there are the values of the outputs for each of the eight color bars in which the image is subdivided, so for every

change in the *bar_cnt* value, a different address from the lookup table is used to generate the outputs. The width of the bar *bar_pix*, used to know if a transition is needed by comparing with the pixel counter, is automatically generated from the number of active pixels, configured in the registers used to create the video timings, using a combinational integer divider by 8. With this technique the color bar is automatically adjusted for every possible resolution and always keeping the correct proportions. The Figure 9.9 shows the state transitions diagrams of this implementation.

```
--White R+G+B
rgb_mem(0) <=   x"EB0" & x"EB0" & x"EB0";
--Yellow R+G
rgb_mem(1) <=   x"EB0" & x"EB0" & x"100";
--Cyan G+B
rgb_mem(2) <=   x"100" & x"EB0" & x"EB0";
--Green G
rgb_mem(3) <=   x"100" & x"EB0" & x"100";
--Magenta R+B
rgb_mem(4) <=   x"EB0" & x"100" & x"EB0";
--Red R
rgb_mem(5) <=   x"EB0" & x"100" & x"100";
--Blue B
rgb_mem(6) <=   x"100" & x"100" & x"EB0";
--Black
rgb_mem(7) <=   x"100" & x"100" & x"100";
--Out of Bars => Black
rgb_mem(8) <=   x"100" & x"100" & x"100";
```

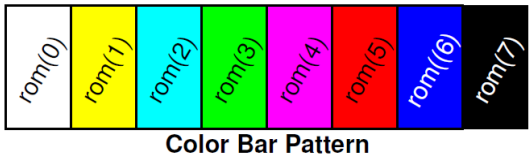


Figure 9.8. Limited range values for every bar stored in ROM lookup table

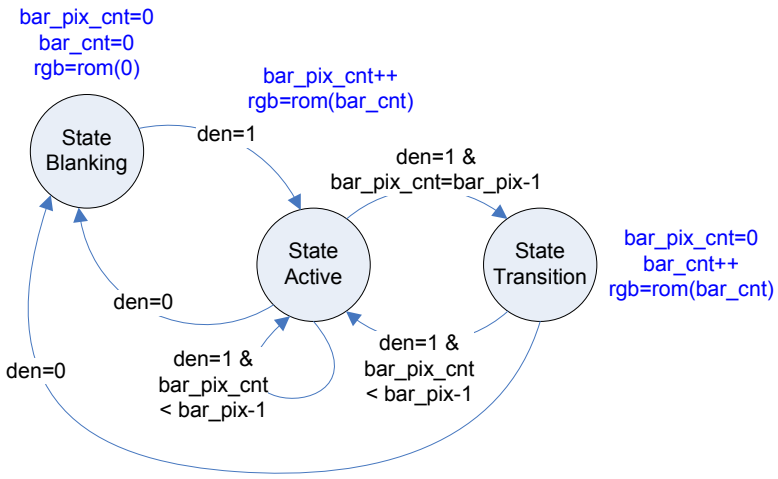


Figure 9.9. State transitions diagram to generate an horizontal color bar pattern

The basic concept to clock data by using states of *den, hsync and vsync* signals can be translated to generate the vertical color bar pattern, although in this case, the lines in a bar are counted instead of the pixels, as the bars are horizontal to the image, and to count it, the *hsync* signal is used to increase the line counter until a bar transition arrives.

9.2.4 Simulation

Some important points have been considered in order to validate that video timings are correctly generated according the CEA-861-E specification. The section is structured with the checkpoint and the respective simulation that determine that it is correctly implemented. All the simulations have been made using Modelsim software.

- All the values configured in registers regarding active/blanking pixels/lines must be correctly used for state transitions depending on the *hcnt* and *vcnt* values. It can be appreciated in all the following figures that state transitions are made according to the configuration registers values, corresponding to 720p video timing. Figure 9.14 and Figure 9.15 show respectively a complete video active line and a complete video frame.

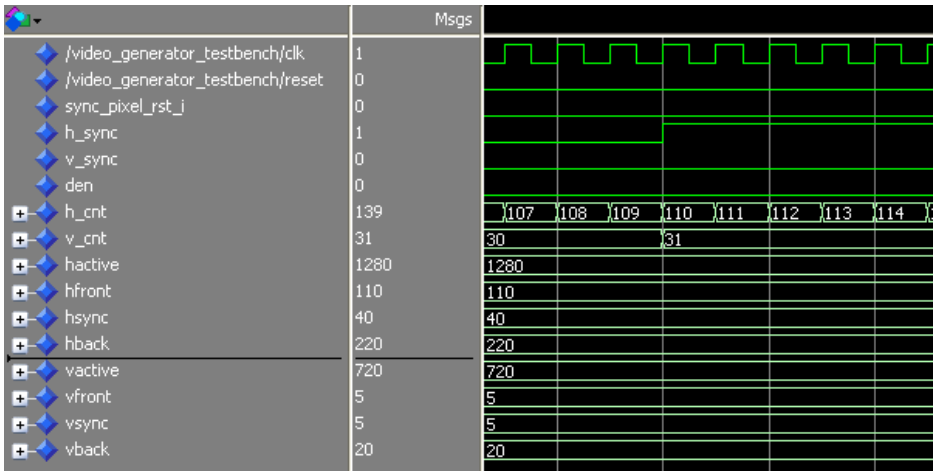


Figure 9.10. Hsync pulse starts after 110 pixel clock cyles (Hfront register)

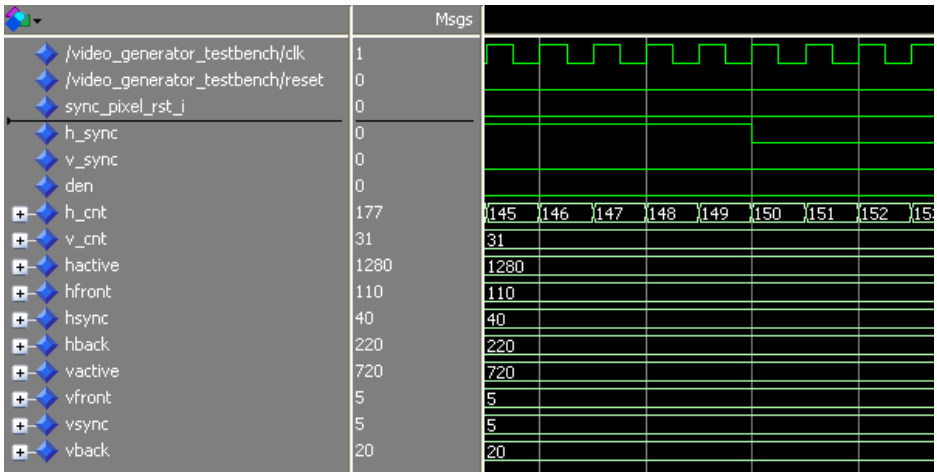


Figure 9.11. Hsync pulse ends after 150 pixel clock cycles (Hfront+Hsync registers)

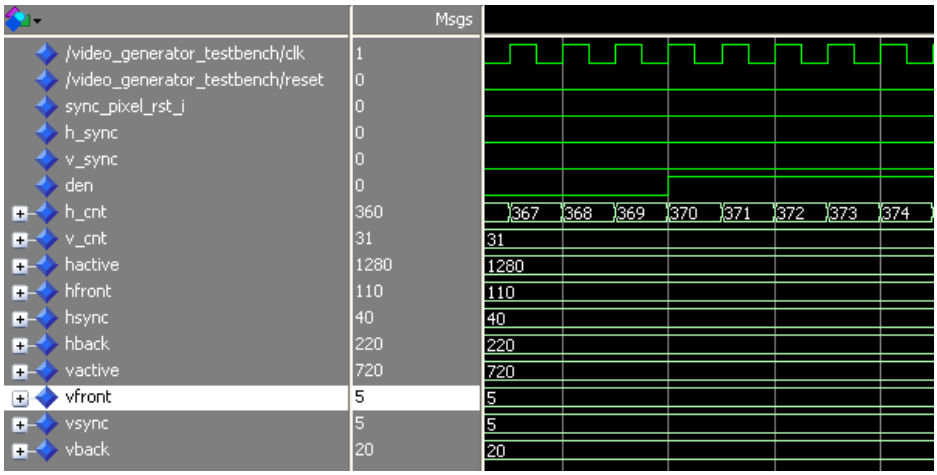


Figure 9.12. Den starts after 370 pixel clock cycles (Hfront+Hsync+Hback registers)

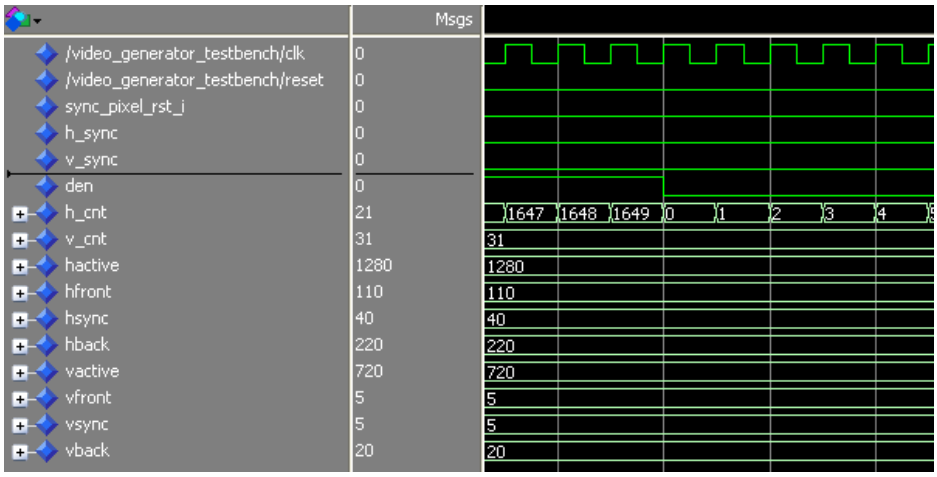


Figure 9.13. Den ends after 1650 pixel clock cycles (Hfront+Hsync+Hback+Hactive registers)

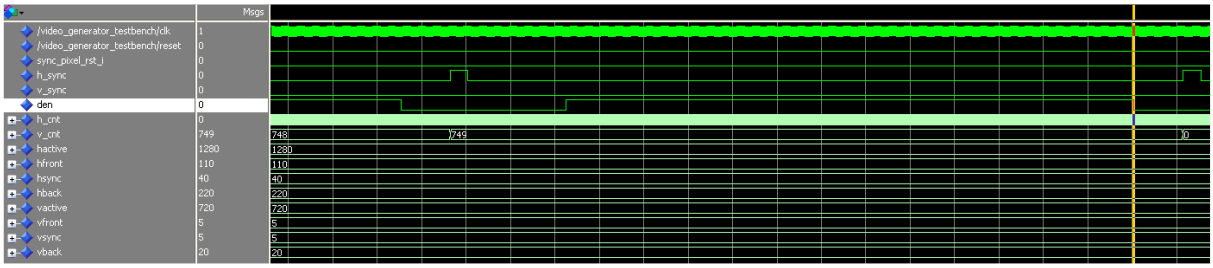


Figure 9.14. 720p progressive video timing video active line waveform

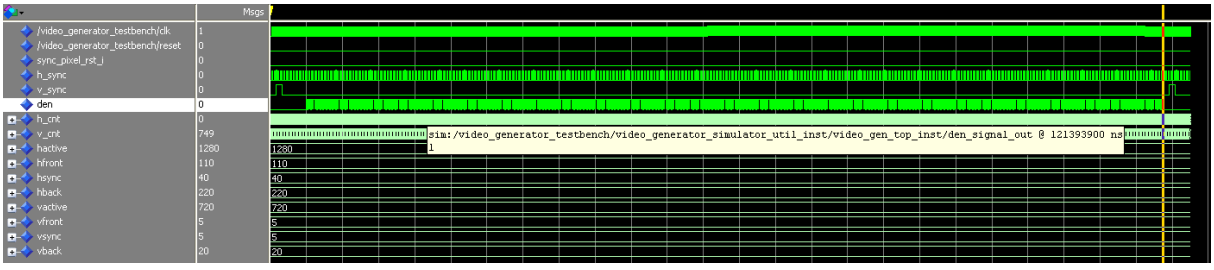


Figure 9.15. 720p progressive video timing complete frame waveform

- The leading edges of *Hsync* and *Vsync* pulses shall be correctly aligned plus or minus zero pixel clocks for progressive video timings and for the first field of interlaced video timings. This behavior is correctly appreciated in Figure 9.16.

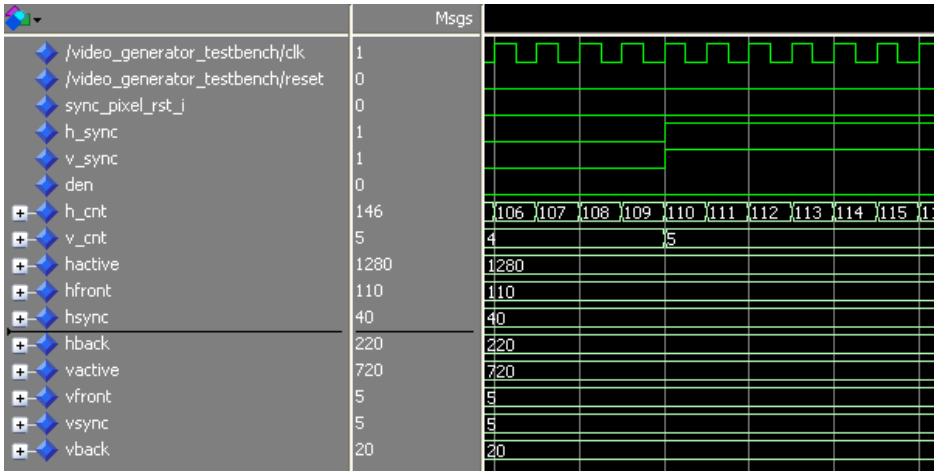


Figure 9.16. Hsync and Vsync pulses alignment

- In the second field of interlaced video timings, the alignment between the leading edge of *Hsync* and *Vsync* transitions shall be precisely a half-line (Htotal/2) plus or minus zero pixel clocks. In Figure 9.17 it can be appreciated the exact alignment as the configuration registers are loaded for the 1080i interlaced video timing. In the first waveform the cursor shows that horizontal counter value is 88 (Hfront) in the Hsync rising edge, and the second waveform shows that horizontal counter value is 1188 (Hfront+Htotal/2) in the Vsync rising edge. For the 1080i video timing Htotal/2=(1920+88+44+148)/2=1100 pixels

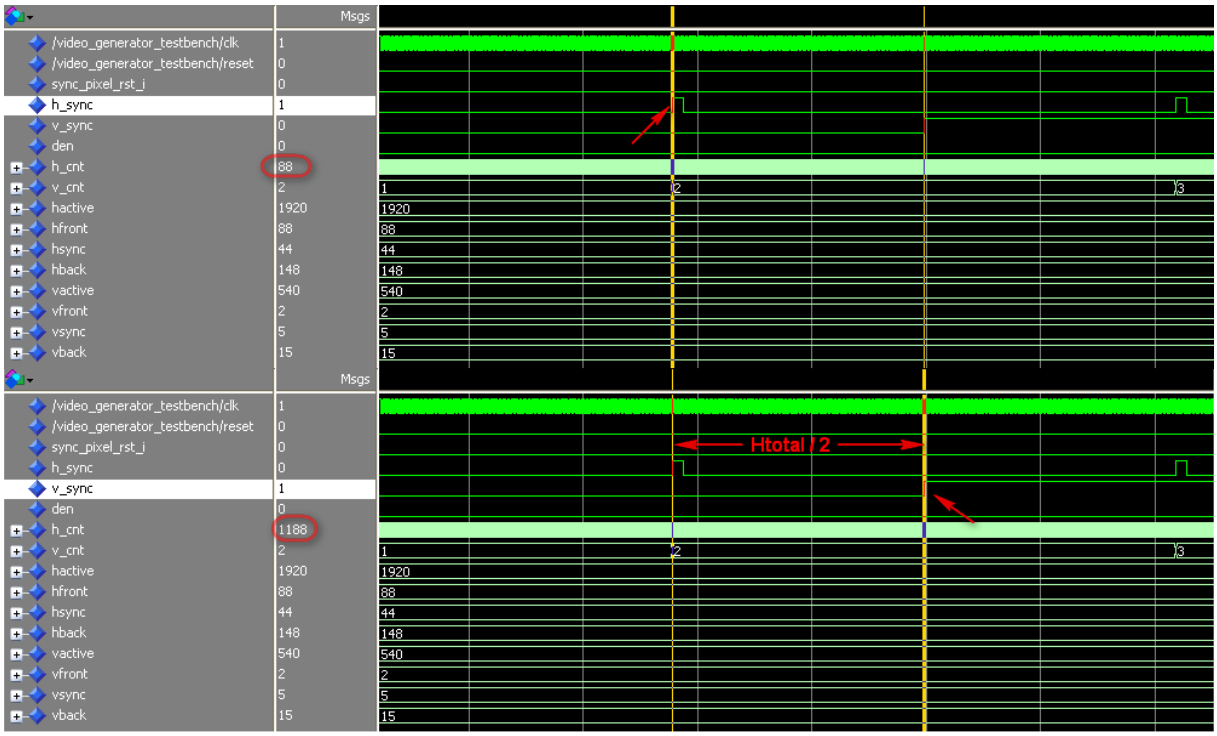


Figure 9.17. Half-line alignment between Hsync and Vsync in 2nd field of interlaced video timings.

- In the even fields of interlaced video timings, the vertical front porch blanking interval must be exactly the value configured in Vfront plus a half-line.
- In the even fields of interlaced video timings, the vertical back porch blanking interval must be exactly the value configured in Vback minus a half-line.
- The conclusion of the last two points is that even fields have an extra blanking line compared with odd fields in interlaced video timings. It can be shown in Figure 9.18 and Figure 9.19 as the extra blanking line is added in the even fieds because the Vsync pulse is separated Htotal/2 pixels from the Hsync pulse.

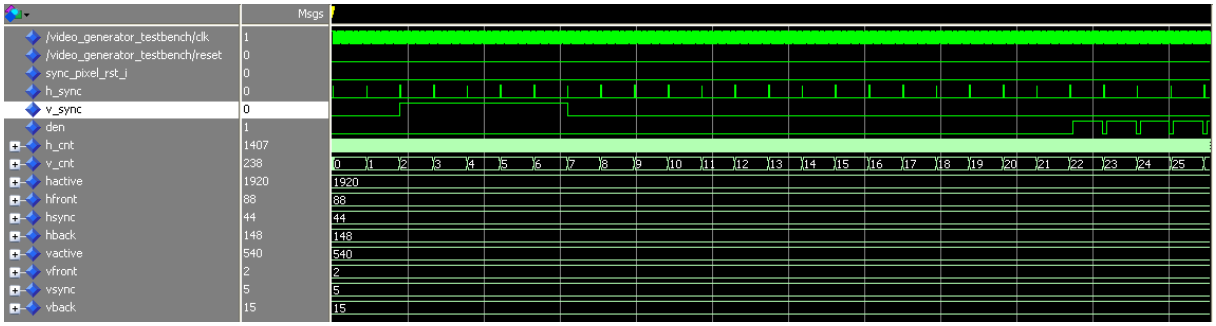


Figure 9.18. Blanking lines (22) in the odd fields of 1080i interlaced video timing

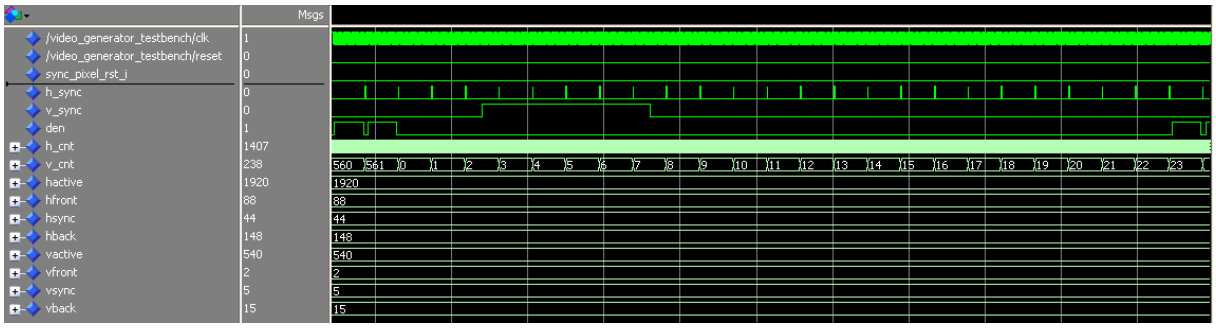


Figure 9.19. Blanking lines (23) in the even fields of 1080i interlced video timing

- In interlaced video timings the active lines are “shared” between the two frames, a half in the odd frames, a half in the even frames.
  In the odd field shown in Figure 9.20, the active period ends after Vfront+Vsync+Vback+Vactive/2 lines. For the 1080i interlaced video timing it corresponds to 2+5+15+540=562 lines.
  In the even field shown in Figure 9.21, the active period ends after Vfront+0.5+Vsync+Vback+0.5+Vactive/2 lines. For the 1080i interlaced video timing it corresponds to 2+0.5+5+15+0.5+540=563 lines.

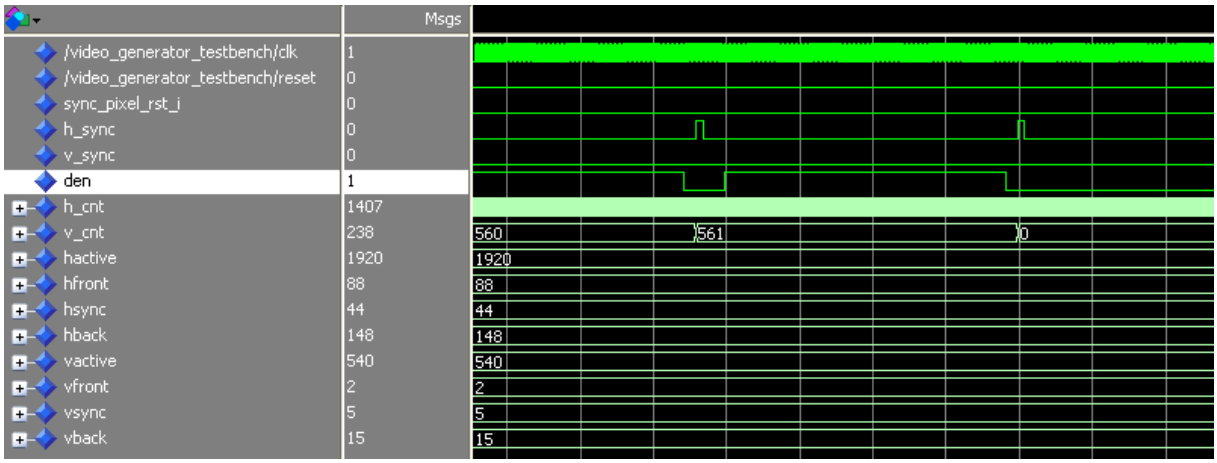


Figure 9.20. Interlaced video timing odd field ends after 562 lines

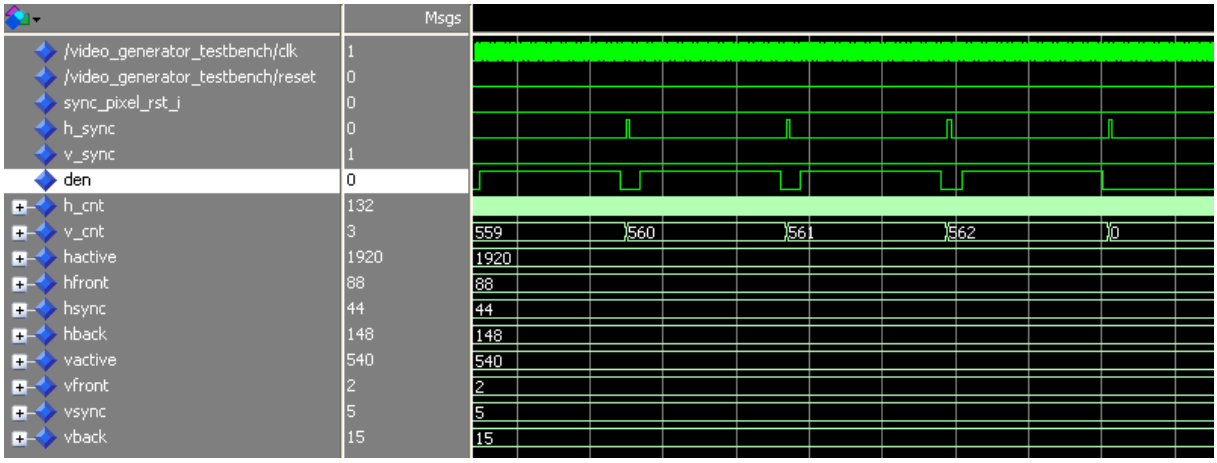


Figure 9.21. Interlaced video timing even frame ends after 563 lines

9.3 Configurable clock dividers block

9.3.1 Introduction

In the previous section about the development of the video generation core it has been explained how the timings are generated by counting the cycles of the master pixel clock. In fact, as CEA-861-E specification explains, the pixel clock rate is also a variable parameter, changing between all the different timings listed in the specification. As the main PLL generates three different video clocks, and another two are generated using constant phase even-number dividers, a clock multiplexer is needed to feed the global pixel clock network with the multiplexer output signal.

The configurable clock divider core is a very simple core with a wishbone slave interface that has different clock division outputs, some fixed and others programmable, and also has a bunch of programmable outputs that can be used to controls the selection lines of the main video pixel clock multiplexer.

The core has been designed as generic as possible in order to ease the options to reuse it in the future. For the prototype system configuration, some output clocks divided by 2 and 4 are needed as it has been seen in the chapter regarding clock network design, so two cores are

instantiated for the video clocks, and another one will be used to generate a variable audio master clock, with the possibility to change the clock rate at run time.

9.3.2 Core internal architecture & hierarchy definition

In the Figure 9.23 it can be appreciated how three instances of the generic configurable clock divider core have been used to generate all the necessary fixed pixel clocks and the configurable audio master clock.
Later this clocks feed the inputs of the video clock multiplexer shown in Figure 9.22 to make the desired pixel clock in the global network selectable at run time. With this approach to change the generated video timing not only the video generation core registers must be changed, but also it has to be assured that the appropriate pixel clock is correctly selected.
In the case of the configurable clock divider for the audio block, a configurable clock division output has been used to be able to directly select the appropriate audio master clock. This clock will be synchronous with the 147,456MHz master reference clock and it will have an exact 50% duty cycle.

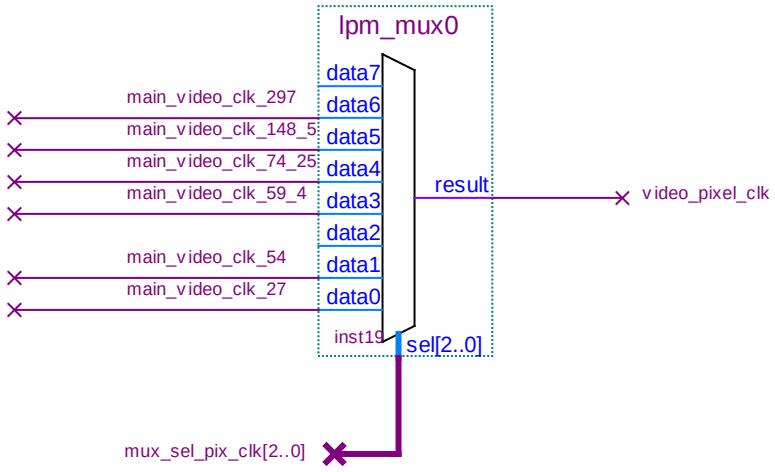


Figure 9.22. Clock multiplexer with pixel clock output connected to global clock network

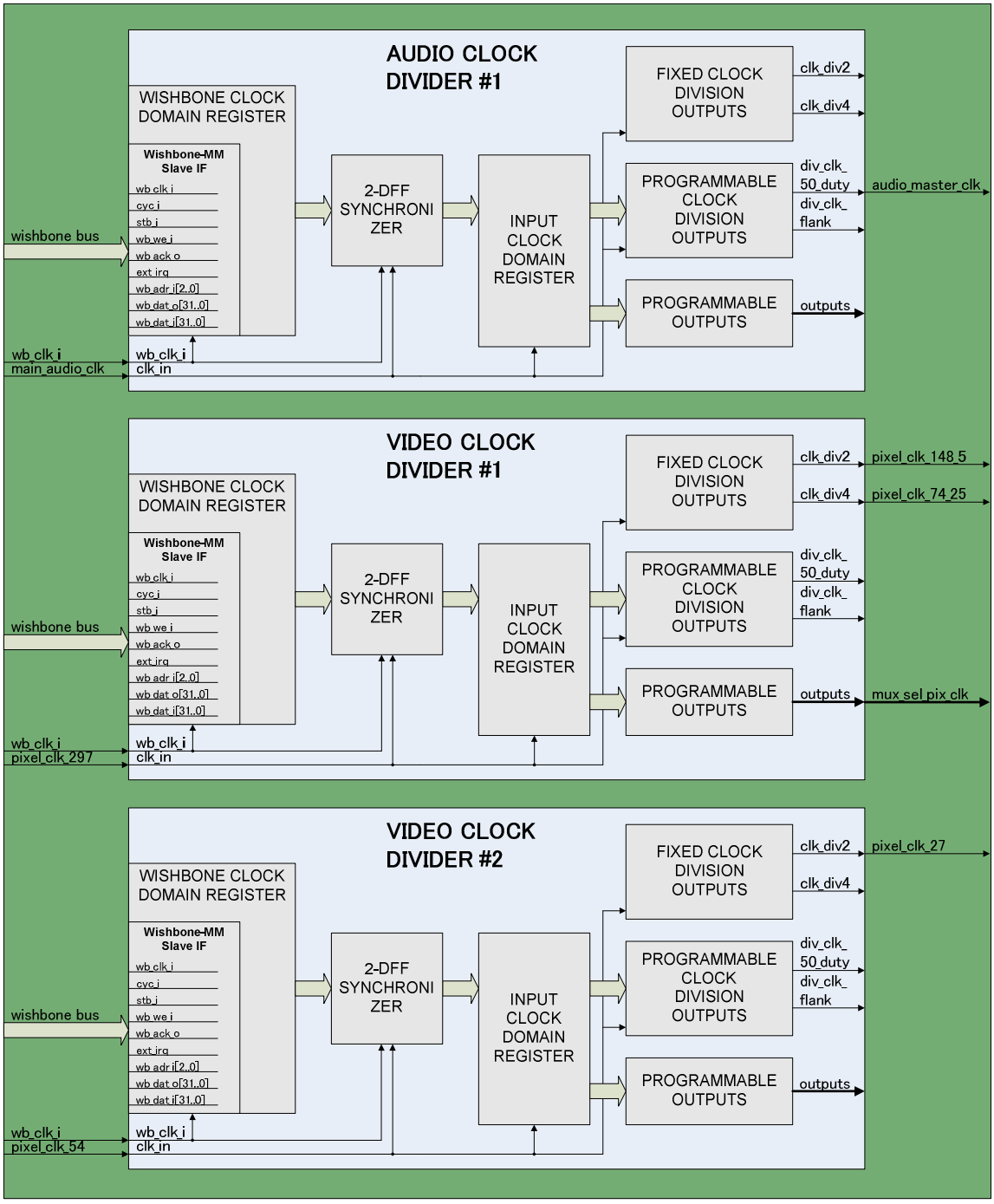


Figure 9.23. Block diagram view of the three configurable clock dividers cores for audio and video generation blocks

9.3.3 Core theory of operation

In this section, the behavior of a single generic configurable clock divider core is described. As it has been explained, not all the clock outputs are finally used in the design but the explanation in this section is oriented to the implementation of the generic core instead of the final integration in the system, that has been already explained before.

As in the previous section the development of a wishbone slave interface has been largely covered, it will not be explained again as the slave interface is based exactly in the same

principles. The registers memory map is included as a reference in *Annex F.3: Configurable Clock Dividers Core.*

The cross-clock domain issue and the possible solutions to this problem, has also been deeply explained in the previous section about the video generation core. Basically in this core the problem also exists for the programmable clock division outputs, and the adopted solution has been the double stage flip-flop synchronizer, as the frequency in wishbone clock domain is always slower than in the other clock domain.

For the fixed clock outputs the typical configuration for a power-of-2 integer division with a simple binary counter has been implemented. The simple 2-bit counter has 4 possible states, and the output clock is the LSB of the 2-bit counter, for the divided by 2 clock output, and the MSB for the divided by 4 output. Such division creates a phase coherent clock with the source clock. Figure 9.24 can be used as an example, as there is a timing diagram to illustrate the same behaviour in a 4-bit binary counter, using as outputs the bits from LSB, source clock divided by 2, to MSB, source clock divided by 16.

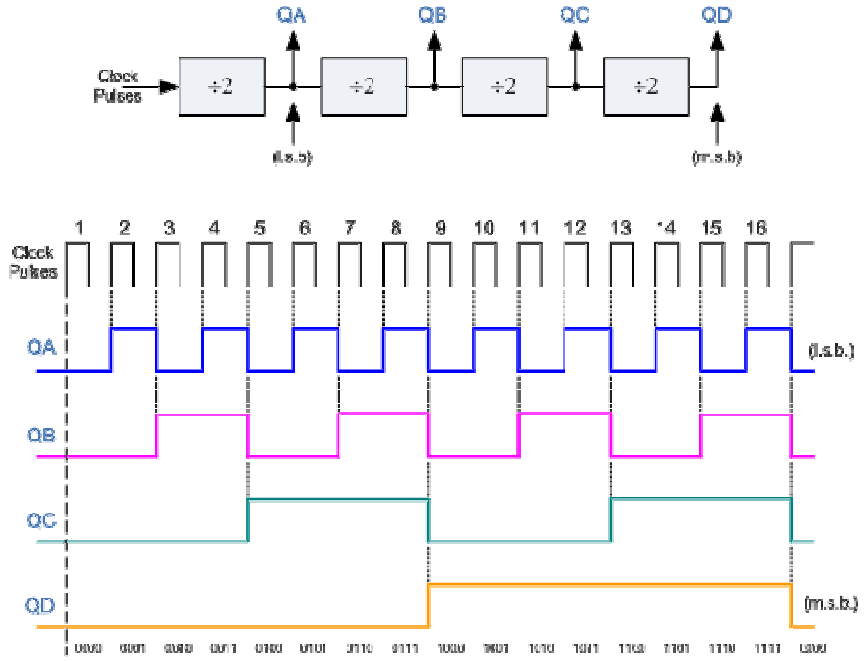


Figure 9.24. Example of frequency division with a 4-bit binary counter

For the 50% duty cycle programmable clock output a similar approach has been used. It consists in a counter from 0 to EVEN_RATIO value configured in the appropriate register. When output count arrives to the end, the enable signal for a T flip-flop goes active during one cycle. That means that every $2\times(1+EVEN\_RATIO)$ input clock cycles, the output of T flip-flop toggles, generating the divided by $2\times N$ even clock dividers. With this implementation this clock is configurable at runtime, by changing EVEN_RATIO value in the associated register, and able to generate perfectly accurate 50% duty cycle output clocks divided by an even integer. See (9.1.) to calculate the output clock frequency.

$$divided\_clk\_50\_duty = clk\_in\frac{clk\_in}{2\times(1+EVEN\_RATIO)} \qquad (9.1.)$$

For the other clock output, able to generate output clocks divided by any integer, odd or even, but with the penalty to have variable duty cycle, the intention is to use the same implementation, but with small modifications.
Basically when the counter from 0 to ODD_RATIO arrives to the end, instead of generating the enable signal for the T flip-flop, the clock output goes directly high. In the next cycle the clock output will go low when the count is restarted from the start. It means that clock outputs are high only during one cycle of input clock, so the duty cycle is completely variable.
Anyway with this trade-off, this approach allows dividers by odd integer numbers, and when the 50% duty cycle is not really needed is better to use this clock output, as obviously it offers more flexibility and it is also configurable at run time by changing ODD_RATIO value in the associated register. See (9.2.) to calculate the output clock frequency.

$$divided\_clk\_flank = clk\_in \frac{clk\_in}{1 + ODD\_RATIO} \qquad (9.2.)$$

9.4 I²S generation

9.4.1 Introduction

As explained in *Chapter 5.3: Proof-of-concept prototype architecture design*, from all the audio input interfaces available in the ADV7511, only the I²S interface is going to be used.

This section explains the development and the general operation of the I²S generation core. In *Annex C.3: Digital Audio Concepts* the basic digital audio concepts are introduced as well as the fundament points of the I²S protocol. These basic concepts can be very useful to complement the core development explanation.

It is important to remark that ADV7511 can only act as a receiver in slave mode, so the I²S transmitter has to be configured as a master, thus generating all the related clocks, and apart from the typical signals present in the I²S protocol, the master clock (MCK) line is mandatory for the ADV7511 I²S interface, and needs to be a multiple of the bit clock (BCK).

In short, this is the list of features of the ADV7511 I²S audio input interface that need to be fulfilled by the I²S generation core:

- It can accommodate from two to eight channels of I²S audio up to a 192KHz sampling rate, so the interface has 4 input data lines.
- All the standard I²S, left-justified and right-justified serial audio formats are supported
- Word length can also be configured between 16 bits and 24 bits.
- To accommodate this sample precision, the accepted bit clock can vary from 32 to 64 times the sampling frequency, and it will be detected automatically.

9.4.2 Core internal architecture & hierarchy definition

In audio test systems, usually pure single frequency tones are used as audio stimulus signals, either in analog or digital form. This kind of test is usually based on acquiring the signal in the DUT audio outputs and compares some of the parameters with the original ones in the stimulus signals. Typically, the basic measurements consist on checking the single tone frequency in the audio outputs is the same than in the original audio signal in the inputs.

The main objective of this core is to connect a digital audio I^2S single tone generator to the input pins of the ADV7511, in order to send this digital audio packetized over the HDMI link and later be able to capture it in the audio outputs of the HDMI sink device under test.
In order to achieve a higher flexibility, the I^2S tone generator does not have a fixed frequency output but it is entirely configurable, it means that the left and right channels in the I^2S data stream can have different frequencies.

The I^2S generation core is built around a modified I^2S transmitter core from Opencores working in master mode. Figure 9.25 shows a block diagram of the core basic architecture and the interaction between blocks. The I^2S transmitter from Opencores is based around two different wishbone slave interfaces:

- Configuration interface. It is used to configure the basic core parameters that define the transmitter operation, as the bit clock rate and the sample clock rate, as well as the audio word length and the behavior of interrupt outputs used to fill the sample buffer at real time.
- Sample buffer interface. It is used to fill in parallel the sample buffer with data that later is serially sent on the I^2S data line. The size of the sample buffer is determined by the Wishbone address bus width and it is addressed by setting the MSB of the address to '1'. The sample buffer is divided in two equal parts, lower and upper, and the core notifies when either is emptied of audio data using two configurable interrupt outputs.

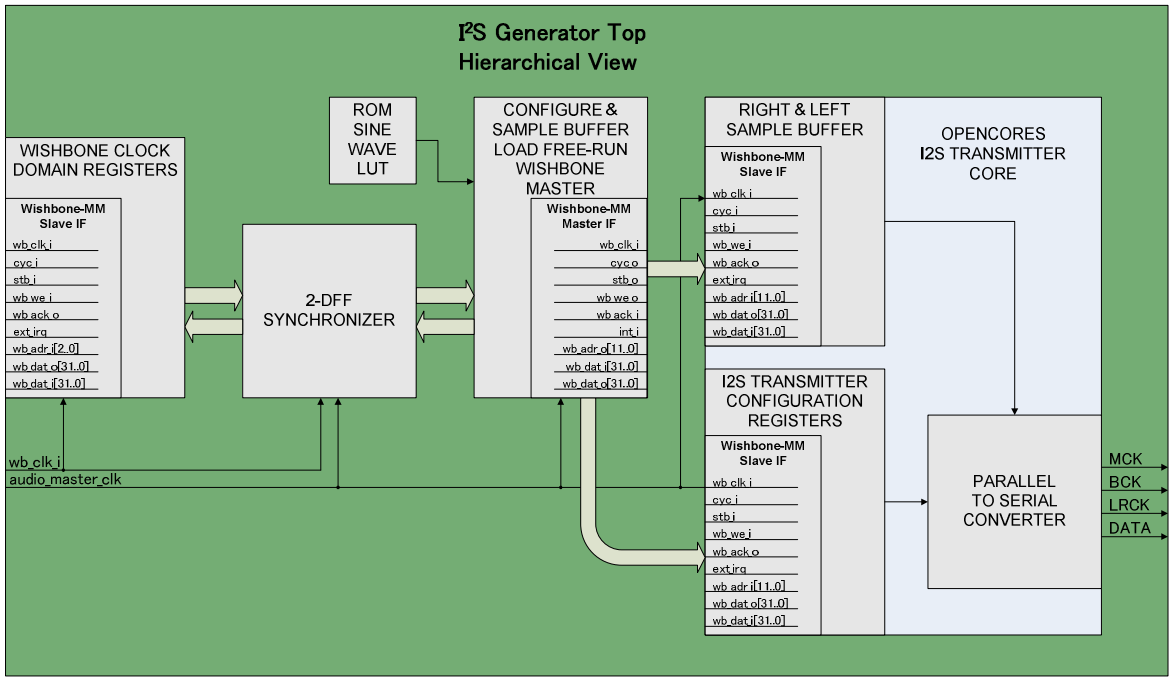


Figure 9.25. Block diagram of core internal hierarchical view

As explained before the audio generator will only produce single tones at specific frequencies and in the time domain, a single tone corresponds to a sine wave of a single frequency. To generate that sine wave in digital samples, a Direct Digital Synthesis (DDS) technique has been used. DDS is a common method to generate periodic or repetitive signals, and is based on storing in a Look-up Table (LUT) some samples of the analog waveform that needs to be created; in this case it stores digital data of a single cycle of a sinusoidal waveform.

Eventually, in order to generate an audio data stream, the sample buffer must be continuously filled with new sample data from the LUT as soon as the older contents in the buffer are

transmitted, so the slave configuration and sample buffer wishbone interfaces are internally connected to a free-running master wishbone interface that cyclically fills the sample buffer with new data and also configures the I^2S transmitter core to the desired values.

It has been discarded to connect directly the I^2S transmitter core to the NiosII master because the transmitter core uses the same wishbone clock to generate I^2S clocks, so it needs to operate at the audio master clock frequency in order to generate a coherent I^2S stream at typical audio bit clock and sampling rates. As it has been explained in the chapter about PLL resources usage & configuration, the main audio clock is running at 147,456MHz.

Anyway, the top hierarchical block of the I^2S generation core it also has a wishbone slave interface running at the system clock frequency in order to change dynamically the core configuration parameters from NiosII. Some of the configurable parameters are simple mirror registers of the transmitter core and others are used to manage the free-running wishbone master and the DDS block.

9.4.3 Core theory of operation

9.4.3.1 Wishbone slave interface implementation & synchronization issues

As in the previous sections the development of a wishbone slave interface has been largely covered, it will not be explained again as the slave interface is based exactly in the same principles. The registers memory map is included as a reference in *Annex F.4: I2S Generation Core.*

The cross-clock domain issue and the possible solutions to this problem, has also been deeply explained in the previous section about the video generation core. Basically in this core the problem also exists because system & audio clocks have variable phase and time relationships. It is also unavoidable that some data needs to be shared between them, so it has to be made with great caution. The adopted solution has been the double stage flip-flop synchronizer, as the frequency in wishbone clock domain 42.5MHz, it will be always lower than in the audio clock domain running at 147,456MHz.

The only extra thing to remark is that an internal signal is asserted after any write transaction, in order to notify the free-running wishbone master to stop the normal operation, and configure again the I2S transmitter core with the new configuration information sent from the NiosII main controller.

9.4.3.2 I^2S Transmitter Core

As it has been explained previously the main low-level block for the I^2S generation is a slightly modified I^2S transmitter core from Opencores. The main modifications are:

- Generation of extra MCK clock needed in the ADV7511 I^2S audio interface. From the ADV7511 documentation *[24][25]*, MCK should take a value related to the other clocks: $128 \times N \times Fs \quad N = 1 \ldots 4$. In order to simplify the configuration, it has been decided to fix the relationship between BCK and MCK to 2x, it means $MCK = 2 \times BCK = 128 \times Fs$
- Better control of number of valid samples in the sample buffer with the generic MAXMEMO.

Another important aspect to remark is the core parameterization made at compilation time using the VHDL generics. The core generics are included as a reference in *Annex F.4: I2S Generation Core.*

- The slave wishbone interface DATA_WIDTH has been configured to 32 bits instead of 16 bits, which has been made to not restrict the core operation to a 16-bit fixed

word length, and be able to test the ADV7511 I^2S interface with different word lengths.

- The slave wishbone interface ADDRESS_WIDTH has been configured to 12 bits. That defines the sample buffer maximum capacity to $2^{ADDR\_WIDTH-1} = 2^{11} = 2048$ samples of 32 bits, that is half of the addressable memory space, as the sample buffer is addressed by setting the MSB bit of the address to '1'.

This basic modifications made in the core do not affect too much the core behavior, and as it has been picked up from Opencores, it is not the objective of this project to explain the core behavior. In contrast, the explanation will be centered on the tricky points about how to use it from the main controller, that in this case it is a cyclic free-running wishbone master interface instead of the NiosII, for the reasons already exposed, avoid the cross-clock domain problem in order to not modify too much the original core and the free-running behavior, that implemented in software it would mean a never-ending task eating CPU all the time to fill the sample buffer cyclically.

9.4.3.3 Sine Wave generation using DDS techniques

Direct digital synthesis (DDS) is a technique to generate a frequency- and phase-tunable output signal, referenced to a fixed-frequency clock source, using a Look-up Table (LUT) that contains digital data of a periodic waveform; in this case a complete cycle of a sine wave is stored in a ROM lookup table.

There are two basic DDS architectures with two different approaches to change output frequency. The first and the simplest architecture is shown in Figure 9.26 and it consists of a LUT implemented using a ROM, and an address counter. As it has been said, the ROM contains 2N samples of a sine wave complete cycle, and the N-bits address counter that increases at every clock cycle, is used to step through and access each of the ROM addresses, presenting the stored contents in the digital output.

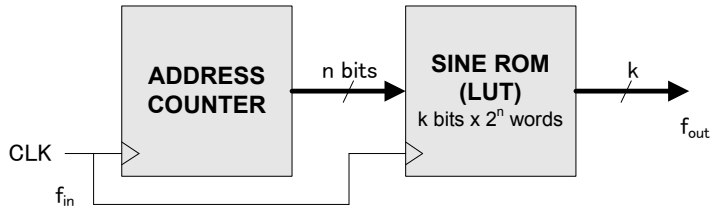


Figure 9.26. Basic DDS architecture

The output frequency (9.3.) of this DDS implementation depends on:

- The source clock frequency, f_{in}
- The number of samples of a sine wave stored in the ROM, $L=2^N$

$$f_{out} = \frac{f_{in}}{2^N} \qquad (9.3.)$$

The big advantage of this approach is the simplicity that offers to generate a fixed output frequency, but the main weakness is that lacks some frequency tuning flexibility because the output frequency can only be changed with a variable source clock or by changing dynamically the contents stored in the ROM, thus complicating the implementation a lot.

The second architecture shown in Figure 9.27 is slightly more complicated and it consists of a LUT implemented using a ROM, and a phase accumulator block.
The phase accumulator block used to replace the simple address counter is based in an n-bit variable modulus counter and a phase register. The increment in the phase register output is determined by the tuning word M and effectively determines how many addresses to skip.

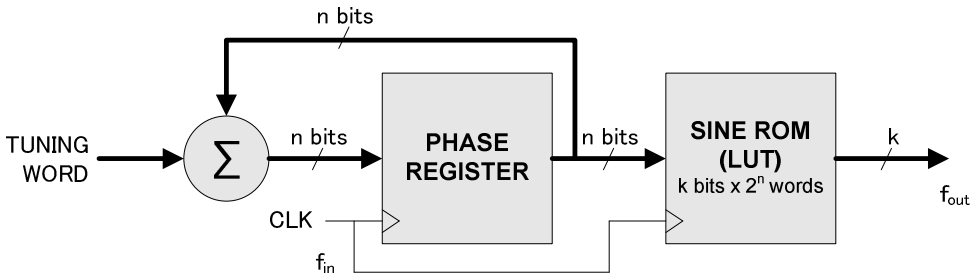


Figure 9.27. Frequency-tunable DDS architecture

The larger the jump size, the faster the phase accumulator overflows and steps through a complete sine wave cycle. A sine wave has a repetitive angular phase range of 0 to 2π, and the counter's carry function acts as a phase wheel, it means it can be represented graphically as shown in Figure 9.28 as a vector rotating around a phase circle, and as the vector rotates around the wheel the sine of the angle generates the output sine wave. Each designated point in the phase wheel corresponds to the equivalent point on a cycle of a sine wave. The tuning word M forms the phase step size and determines how many points to skip around the phase wheel in every jump. If the M value is changed to 0111...1111, the phase accumulator will overflow after only 2 reference-clock cycles (the minimum required by Nyquist).

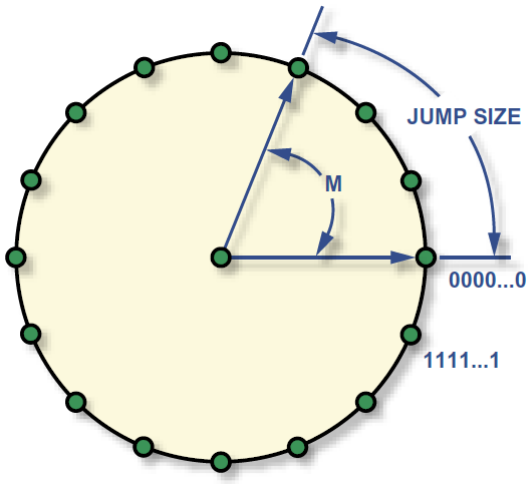


Figure 9.28. Digital phase wheel

The output frequency (9.4.) of this DDS implementation depends on:

- The source clock frequency, f_{in}
- The number of samples of a sine wave stored in the ROM, $L=2^N$
- The binary tuning word that determines the phase jump size, M

$$f_{out} = \frac{M \times f_{in}}{2^N} \qquad (9.4.)$$

The big advantage of this approach is that while keeping the simplicity of the first architecture, the tuning flexibility is higher, because the output frequency can be changed easily while keeping a fixed source clock and fixed contents in the ROM. Changes in the value of M in the DDS architecture result in immediate and phase-continuous changes in the output frequency.

A short basic explanation of DDS architectures has been made to illustrate the implementation main decisions, but for a further explanation some references are provided *[45] [46]*

In an FPGA is quite easy to implement any of the introduced DDS architectures, but the second architecture has been chosen because of the output frequency configurability that offers by only changing the tuning word value (M) that can be easily stored in a register.

Usually in a complete DDS solution, the time-varying signal in digital form is directly connected to a D/A converter to generate analog waveforms, but in the I^2S generation core, the operation is slightly different. As it has been explained in the introduction, the data for each of the two channels is serially sent in the DATA line after a transition in the LRCK clock signal. The LRCK clock runs at the sampling rate (Fs), and that means that a new sample is required after every Fs cycle. In fact the intermediate sample buffer of the final I^2S transmitter block is not filled after every Fs cycle, but in order to calculate the frequency of the digitized sine wave that is serially sent in the I^2S stream, the sample rate frequency must be used as the effective source clock. The ADV7511 accepts a vast range of sampling rates up to 192KHz, so the maximum supported sampling rate will be used to constrain the design and select:

- The modulus number where the phase accumulator overflows.
- The LUT number of samples stored in the ROM.

In order to keep resources to the minimum and trying to fulfill the criteria of an optimum output frequency resolution, a ROM with L=1920 samples have been chosen and the phase accumulator will also overflow at 1920. The output frequency resolution can be counted with the minimum possible jump size of M=1. Equation (9.5.) shows that this selection offers a good output frequency resolution of 100Hz at the maximum sampling rate.

$$f_{res} = \frac{M \times f_{in}}{L} = \frac{1 \times 192k}{1920} = 100Hz \qquad (9.5.)$$

The Nyquist Theorem dictates that there is a minimum of two samples per cycle required to reconstruct the desired output waveform without aliasing effects and that means that the jump step size M range is limited from 1 to 960. The theoretical maximum achievable output frequency $f_{out} = 96kHz$ is far from the generally accepted range of audible frequencies that goes form 20Hz to 20kHz. Table 9.2 shows the main the output frequency resolution and range for different sampling rates that can be configured in the I^2S transmitter with the current audio master clock, and that are accepted by the ADV7511 I^2S interface.

Sampling Freq. (Fs)	Output Frequency Resolution (f_{res})	Output Frequency Range (f_{out})
32 kHz	16,66Hz	[16,66Hz-16kHz]

48 kHz	25Hz	[25Hz-24kHz]
96 kHz	50Hz	[50Hz-48kHz]
192 kHz	100Hz	[100Hz-96kHz]

Table 9.2. Output frequency range and resolution for different sampling rates

Finally to calculate the ROM sample values, first of all the angular resolution for a LUT with L memory words is needed (9.6.). With the angular resolution in the phase wheel the values for the normalized sine wave LUT without offset can be calculated (9.7.).

$$\Delta\theta = \theta(i) - \theta(i-1) = \frac{2\Pi}{L}, \quad i = 0 \ldots L-1 \qquad (9.6.)$$

$$LUT(i) = \sin\left(\frac{2\Pi}{L} \times i\right), \quad i = 0 \ldots L-1 \qquad (9.7.)$$

The selected ROM will have 1920 samples with 16-bits depth, as 16 bits has been considered enough to represent the sine wave with proper accuracy. The 16 bits allow 2^{16} (65536) quantization levels and the analog LUT range goes from 0 to 2. The digital values for every sample have been finally calculated in order to use the full dynamic range (9.8.). As the sine wave does not have an offset applied the sample values are signed, and with a 16 bits depth the sample range is -32768..32767 with a center point of zero. The sign encoding used in the binary conversion is two's complement.

$$LUT(i) = \sin\left(\frac{2\Pi}{L} \times i\right) \times \frac{2^{16}-1}{2}, \quad i = 0 \ldots L-1 \qquad (9.8.)$$

The Figure 9.29 shows some lines of the final Memory Initialization File (.mif) *[47]*, an ASCII text file that specifies the initial content of the ROM memory block instantiated with the QuartusII ROM megafunction.

The final Memory Initialization File has 1920 words of 16-bits width, and Figure 9.29 also shows the addresses that correspond to some singular points of the sine wave.

```
-- Quartus II generated Memory Initialization File (.mif)
-- 1920 mostres d'una sinusoide codificades en 16 bits.
-- Centrada en zero i sense offset.
-- Negatius amb complement a 2

WIDTH=16;
DEPTH=1920;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
        000  :   0000;
        001  :   006B;
        002  :   00D6;
        003  :   0141;
        004  :   01AC;
        005  :   0218;
        006  :   0283;
        007  :   02EE;
        008  :   0359;
        009  :   03C4;
        00A  :   0430;

        1DD  :   7FFD;
        1DE  :   7FFE;
        1DF  :   7FFF;
        1E0  :   7FFF;
        1E1  :   7FFF;
        1E2  :   7FFE;
        1E3  :   7FFD;
        1E4  :   7FFC;

        3BD  :   0141;
        3BE  :   00D6;
        3BF  :   006B;
        3C0  :   0000;
        3C1  :   FF95;
        3C2  :   FF2A;
        3C3  :   FEBF;
        3C4  :   FE54;

        77D  :   FEBF;
        77E  :   FF2A;
        77F  :   FF95;
END;
```

Addr 0x000 (0, 2π)

Addr 0x1E0 (π/2)

Addr 0x3C0 (π)

Figure 9.29. Memory Initialization File for the sine wave samples ROM

The explanation of how the I^2S transmitter sample buffer is filled, as well as the phase accumulator block implementation is tightly bound with the free-running Wishbone master interface, that is fully explained in the next subsection.

9.4.3.4 Free-running Wishbone master interface implementation

For the already explained reasons, the free-running wishbone master is in charge of constantly filling the sample buffer with the ROM sine wave samples and also configures the I^2S transmitter core when some configuration change is ordered from the NiosII main controller.

The high-level operation of this block will be explained using the state transitions diagram found in the Figure 9.30 where the cyclical process can be clearly appreciated.

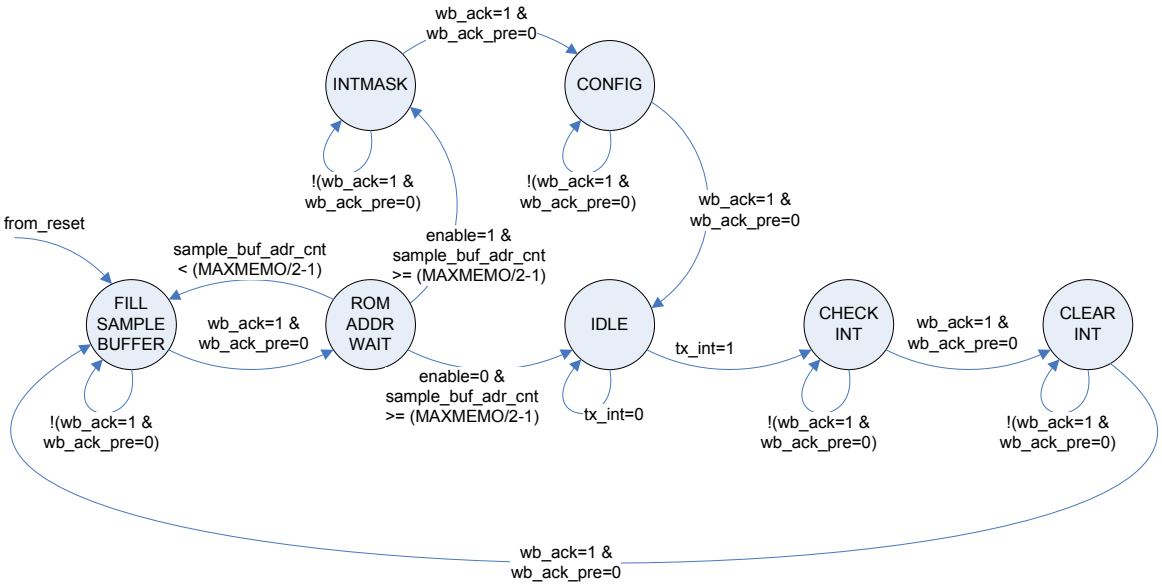


Figure 9.30. State transitions diagram for the free-running wishbone master interface

The condition checked for state transitions is usually the completion of a read/write transaction from the master to the I^2S transmitter core. As it has been explained in the *Chapter: 8: Hardware Architecture*, the master is notified to end the transaction when the slave asserts the handshaking signal *wb_ack.*

First of all, the core starts in a known state after reset or any update on the registers of the slave interface from NiosII. That means that every write transaction from NiosII generates a reset for the internal free-running wishbone master, which will halt the normal operation to configure again the I^2S transmitter operation from scratch. The cyclical process to configure the I^2S transmitter is summarized showing the evolution between the different states shown in the transitions diagram.

Fill Sample Buffer State + Rom Addr Wait State: cyclically filling sample buffer
After a reset, the core starts to fill the two equal parts of the sample buffer. The internal signal *sample_buf_adr_cnt* is increased every time a write transaction is completed until it arrives to MAXMEMO/2, meaning that one half of the buffer is completed and the signal *sample_buf_adr_offset* is used to address one part or the other. The phase accumulator block, used to get a value from the ROM to write it in the sample buffer, is implemented by adding the value JUMP_COUNTER_SAMPLES_XX to the modulus counter signals *cont_left* or *cont_right* that are increased every time a write transaction is completed, and used to update the signal *rom_adr_o* to correctly address the ROM and get the sample value in the registered ROM output. Just when the wishbone write transaction is completed, the *rom_adr_o* signal is already updated to get the next sample value. As the synchronous ROM output is registered and it needs one cycle after address is changed to get the value in the output, an additional state Rom Addre Wait is used to wait until the ROM output has the correct sample value that will be written in the sample buffer in the next cycle.

The core makes this write operation cyclically until a part of the buffer is fully completed, checking if *sample_buf_adr_cnt* signal exceeds half of the sample buffer space, and depending on whether it is the first state after a reset or not, the core goes to the INTMASK state, starting to configure the I^2S transmitter operation, or to the IDLE state, waiting until an interrupt notifies than one part of the buffer is already emptied and need to be filled again.

In order to show this operation with a timing diagram and taking advantage that all the operations are completed in a short timeframe, the Altera SignalTap II logic analyzer has been used to take a capture, of all the internal and bus signals involved, to illustrate the evolution of signals while the core remains in the Fill Sample Buffer state.

Figure 9.31 shows the capture that has been made using the reset release as the acquisition trigger. The four cycles needed to complete the write operation can be clearly appreciated.

0. After a reset the core starts in the Fill Sample Buffer state and *rom_addr_o* points to the first ROM address, so data from the first position is already available
1. Master initiates the transfer by addressing to the first sample buffer address *wb_adr_o=0x800*, presenting the valid sample from ROM *wb_dat_o=0x00000000* and asserting *wb_cyc_o, wb_stb_o* and *wb_we_o* signals.
2. Slave sample buffer interface capturs the valid data and immediately asserts *wb_ack_i* signal. The wishbone master detects the rising edge by checking the signal current state and last state: *wb_ack_i='1'* and *wb_ack_i_pre ='0'*
3. The address counters *cont_left, cont_right* are updated depending on the *lr_toggle* variable, because the master subsequently fills one sample for the left channel and another for the right channel. The two counters are increased with the configured jump size, making possible as explained to step through the ROM sine wave samples at different speed. In this case the *cont_left* is updated and the *cont_right* value is assigned to *rom_adr_o* in order to have the sample for the right channel available in the next signal. When the operation is the contrary, for example in cycle #7, the *cont_right* signal is updated and the *cont_left* value is assigned to *rom_adr_o.* All this operation is made in an extra cycle, because it needs one cycle to have the valid data from ROM after presenting the address, and data must be valid before send it to the sample buffer. An additional state Rom Addr Wait is used to achieve it.
4. Finally the core returns to the Fill Sample Buffer state and the *sample_buf_adr_cnt* signal is updated to address to the next sample buffer memory space.

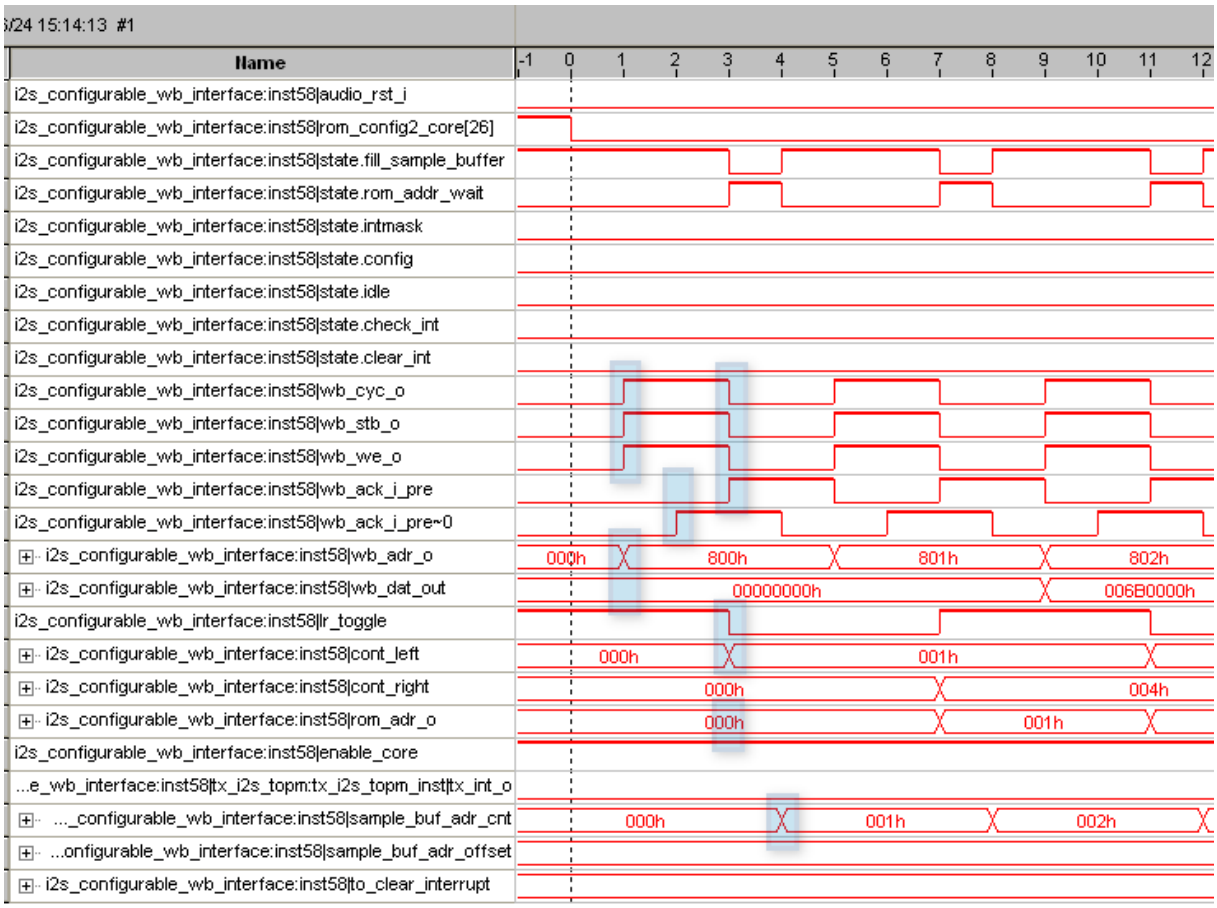


Figure 9.31. Timing diagram during FILL SAMPLE BUFFER state just after a full reset

This process, completed in 4 cycles, to get the valid data from ROM and fill the sample buffer is made until one half of the buffer is completely full. In the timing diagram can be clearly appreciated how *cont_left* and *cont_right* variables are updated with different jump size values, one by one for the left channel and four by four for the right channel, and how sample buffer is subsequently filled with one sample for left channel and another for right channel, exactly the same order the samples are sent in the I^2S Data line. It is also remarkable than although the ROM has a 16-bit depth, the *wb_dat_o* width is 32 bits, so the 32 bits sample is generated with the ROM data in the MSB.

Figure 9.32 shows the capture that has been made using the start of the Idle state as a trigger. It can be clearly appreciated how the core leave the Fill Sample Buffer state when *sample_buf_adr_cnt* arrives at the end of the first half of the sample buffer space MAXMEMO/2-1=959 (0x3BF). This is the normal state transition when the signal *enable* is deasserted and the sample buffer is constantly filled when needed.

Figure 9.32. Timing diagram of core signals when leaving FILL SAMPLE BUFFER state

Intmask State + Config State: configuring the I²S Transmitter core only after a reset
After a reset the core fills the first half of the sample buffer with data from ROM, but the *enable* signal remains asserted until a complete configuration is made, so only in that case, the core does not go to the Idle state when it leaves the Fill Sample Buffer state.

There are two configuration registers in the I²S Transmitter core that needs to be initialized before enabling the core to start the I²S continuous data stream *[39]*. The first register is the core main configuration register TxConfig that is initialized from the mirror register configurable from NiosII, and the other is the TxIntMask register that is initialized with a fixed value that means that interruptions are generated always when one half of the sample buffer is empty.

In order to show this configuration with a timing diagram and taking advantage that all the operations are completed in a short timeframe, the Altera SignalTap II logic analyzer has been used again to take a capture, of all the internal and bus signals involved, to illustrate the evolution of signals and the state transitions.

Figure 9.33 shows the capture that has been made using the start of the Intmask state as a trigger. The six cycles needed to complete the configuration can be clearly appreciated.

0. When core leaves the Rom Addr Wait state it does not return to Fill Sample Buffer because half of the sample buffer memory space is already filled. It goes to Intmask state because *enable* signal is asserted.

1. Master initiates the transfer by addressing to the TxIntMask register *wb_adr_o=0x002*, presenting the fixed value *wb_dat_o=0x00000003* and asserting *wb_cyc_o, wb_stb_o* and *wb_we_o* signals.
2. Slave configuration interface captures the valid data and immediately asserts *wb_ack_i* signal. The wishbone master detects the rising edge by checking the signal current state and last state: *wb_ack_i='1'* and *wb_ack_i_pre ='0'*
3. The core leaves the state Intmask to go to state Config.
4. Master initiates the transfer by addressing to the TxConfig register *wb_adr_o=0x001*, presenting the default value from the mirror register *wb_dat_o=0x00200403* and asserting *wb_cyc_o, wb_stb_o* and *wb_we_o* signals. With this configuration information, the I^2S transmitter core is enabled and starts the I^2S data stream with the samples already present in the sample buffer
5. Slave configuration interface captures the valid data and immediately asserts *wb_ack_i* signal. The wishbone master detects the rising edge by checking the signal current state and last state: *wb_ack_i='1'* and *wb_ack_i_pre ='0'*
6. The core leaves the state Intmask to go to state Idle, waiting for an interrupt that notifies that one of the two halves of the sample buffer is empty and needs to be filled again.

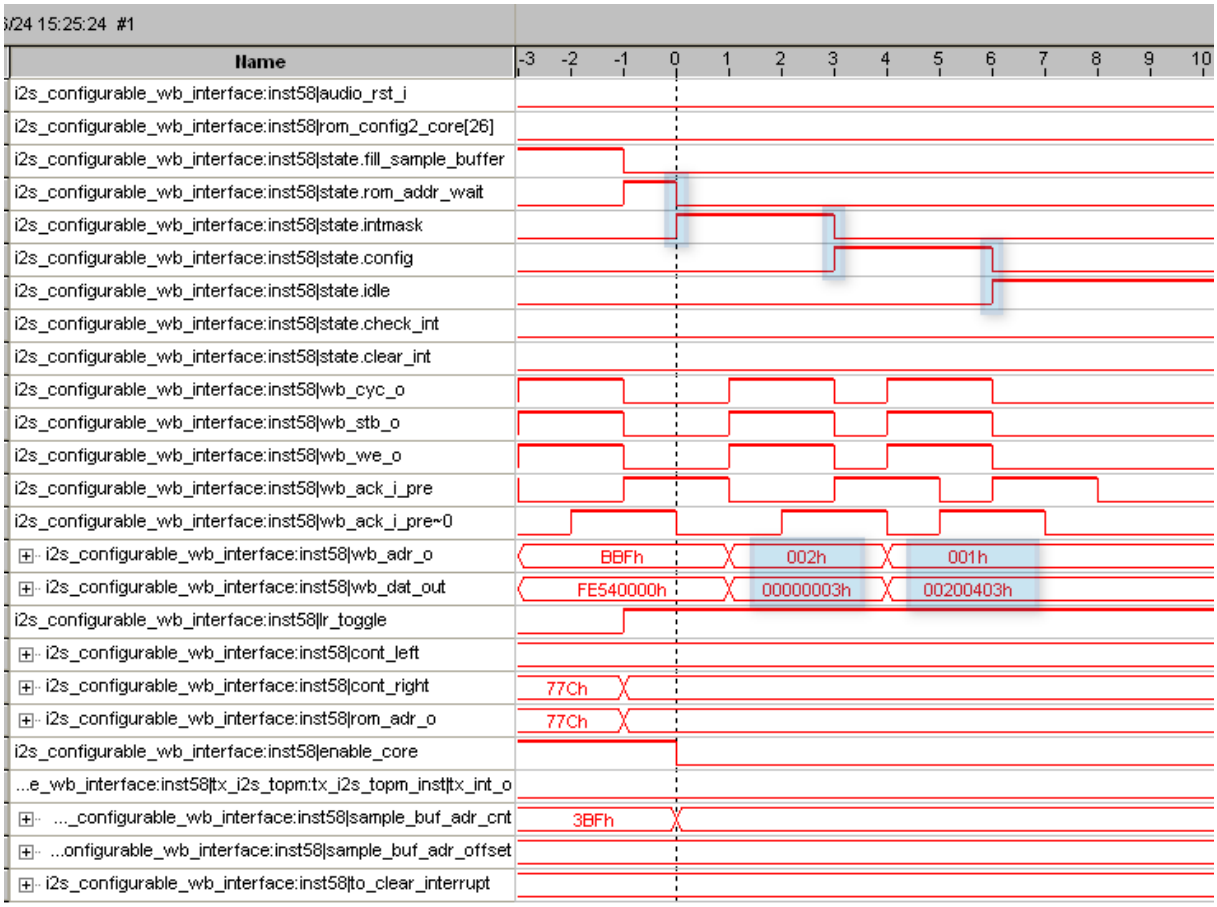


Figure 9.33. Timing diagram of core signals and state transitions during initial configuration

Check Int State + Clear Int State: checking interrupt source and clearing it
When the sample buffer has been filled and the I^2S Transmitter core is already configured, the core remains in the Idle state just checking the interrupt output that the I^2S Transmitter core uses to notify that a half of the sample buffer needs to be filled again.

When the interrupt is received the free-running master checks which part of the buffer needs to be filled by reading the TxIntStat register, as a bit in this register is set to '1' when an event occurs. A logic high value '1' needs to be written in the same bit to clear the interrupt signal.

In order to show how the master core deals with the interrupt condition and clears it with a timing diagram, the Altera SignalTap II logic analyzer has been used again.

Figure 9.34 shows the capture that has been made using the start of the Check Int state as a trigger. The six cycles needed to read the interrupt source and clear it can be clearly appreciated.

0. The core leaves the Idle state and goes to the Check Int state because interrupt signal *tx_int_o* is asserted during the previous cycle
1. Master initiates a read transfer by addressing to the TxIntStat register *wb_adr_o=0x003*, and asserting *wb_cyc_o* and *wb_stb_o* signals.
2. Master captures the valid data presented by the slave configuration interface in *wb_dat_in* signal when it detects the rising edge by checking the signal current state and last state: *wb_ack_i='1'* and *wb_ack_i_pre ='0'*.
3. The core leaves the state Check Int to go to state Clear Int. The signal used to address one of the two parts of the sample buffer *sample_buf_adr_offset* is updated depending on which of the two parts has generated the interrupt and the signal *to_clear_interrupt* get the previously read value that is used to clear the interrupt.
4. Master initiates the transfer by addressing to the TxIntStat register *wb_adr_o=0x003*, presenting the value from the signal *to_clear_interrupt* in *wb_dat_o=0x00000001* and asserting *wb_cyc_o, wb_stb_o* and *wb_we_o* signals.
5. Slave configuration interface captures the valid data and immediately asserts *wb_ack_i* signal. The wishbone master detects the rising edge by checking the signal current state and last state: *wb_ack_i='1'* and *wb_ack_i_pre ='0'*
6. The core leaves the state Clear Int to go to state Fill Sample Buffer and restart the cyclical operation again. With the previously written value, the I^2S transmitter core clears the interrupt signal *tx_int_o*.

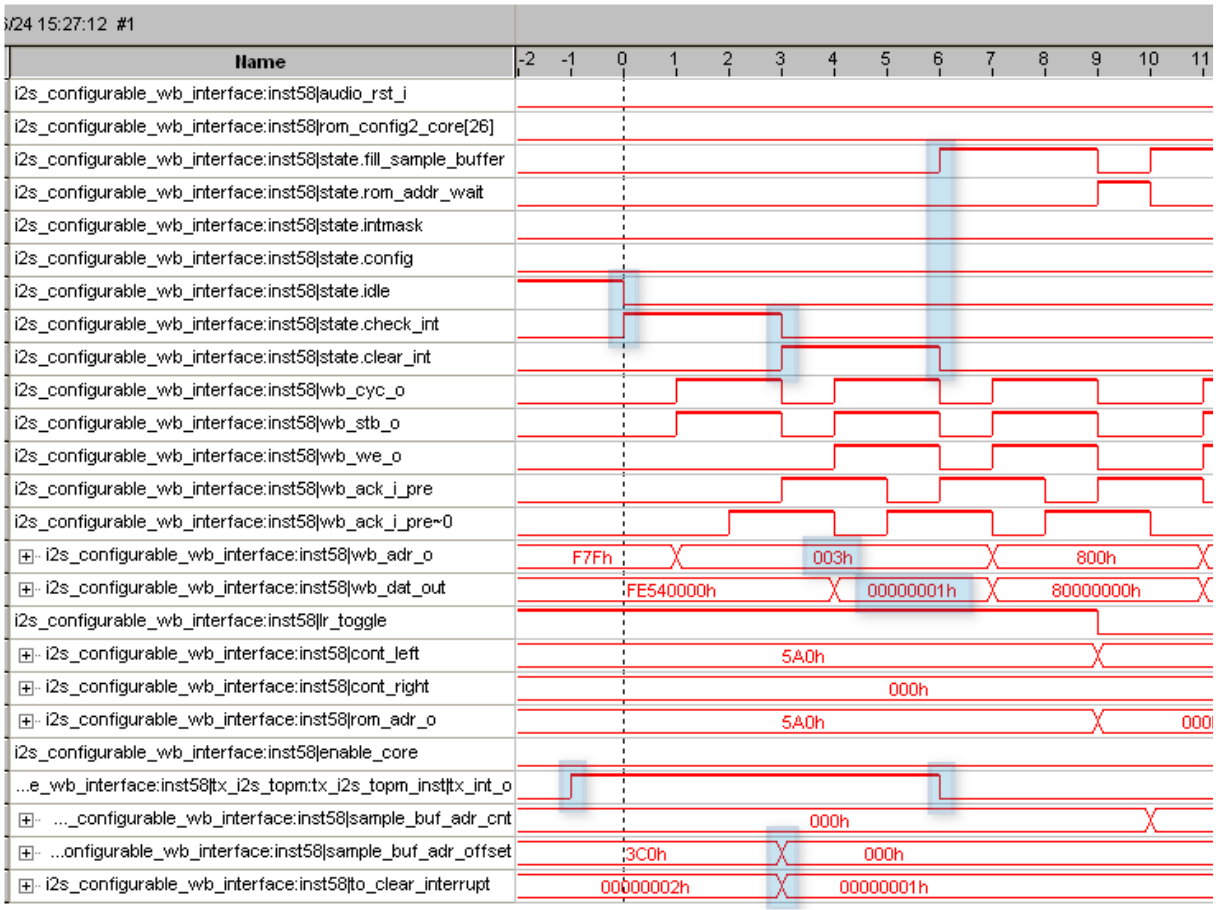


Figure 9.34. Timing diagram of core signals and state transitions during interrupt handling

9.5 HW interrupts & reset control

9.5.1 Introduction

Although all the main system blocks are connected to main controller buses, either Avalon or Wishbone, sometimes external asynchronous interrupts are needed in order to notify the main controller that some block into the system needs attention.
In the prototype, the main I^2C master block has an external interrupt output that can be used for the core control driver that will be implemented in the main controller, and the ADV7511 transmitter also has an external interrupt line itself, that is thought to ease the software developer task as it can be configured to notify several of the transmitter states, and in some way reduce the need of I^2C monitoring traffic at defined intervals between the main controller and the ADV7511. As software is going to run a RTOS with multitasking capabilities, this interrupt is very useful, and enables the possibility to implement an interrupt-driven system controller.

There are two possible approaches to handle interrupt requests (IRQ) from the system controller:

- Using Nios II interrupt controller with IRQ lines directly connected to the microprocessor.
- Registering hardware interrupts from another core directly connected to Wishbone bus.

Any of these two different approaches will be evaluated measuring the response time, which is the time needed to begin executing code specific to the exception cause.
The decision is usually made from software point of view to minimize the response time. In the first case, after detecting the interrupt, the RTOS has to save the registers into the stack to be able later to restore the context, determine the exception source and transfer the control to the adequate interrupt service routine (ISR) for that exception.
In the second case, a software task is constantly polling the register value to detect if an interrupt occurs, to later transfer the control to another task.

From hardware point of view is difficult to decide which is the best method, so both methods has been kept in order to be able to evaluate response time from software point of view to decide which one is better.

That decision means that the interrupts need to be registered to be able to read from the master if a transition has appeared, and also that the bus *irq* signal needs to be or-wired from all the interrupt signals from cores, in the particular case of the HDMI generator board, only the external signal from ADV7511 and the interrupt signal from the I^2C Master Core

Another important aspect in the system is the necessity to reset some cores from the NiosII controller. Sometimes it can be better to start registers configuration in the system cores from a known state, and a selectable reset applied to only a small block of the system can be used frequently in the controller software.

9.5.2 Core internal architecture & hierarchy definition

In the Figure 9.35 it can be appreciated the simplicity of this core, very similar to a programmable input/output. The interrupts from other cores or from external peripherals are synchronized with the system wishbone clock using a double-stage synchronizer to avoid possible cross clock domain problems. In the case of the interrupt signal from ADV7511, this signal is usually triggered when an HDMI sink is detected, so it is clear that is an asynchronous external interrupt. Finally the value in the associated register is directly presented in the reset outputs.

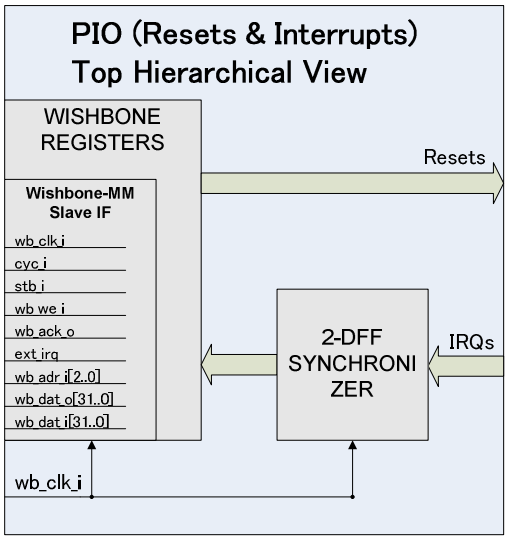


Figure 9.35. Block diagram of core internal hierarchical view

9.5.3 Core theory of operation

As in the previous sections the development of a wishbone slave interface has been largely covered, it will not be explained again as the slave interface is based exactly in the same principles.
The registers memory map is included as a reference in *Annex F.5: HW Interrupts & Reset Control Core*

For the reset outputs, the value of the internal register is directly loaded to the outputs during a fixed time of 10 system clock cycles, approximately 230 ns. After this time the active reset signal is automatically deasserted, so the core that has been reset can restart its proper operation.

For the interrupt outputs a classic double-stage FF synchronizer is used to avoid problems if the interrupt source is asynchronous with the wishbone clock, and that is the case for any interruption coming from human interaction, for example the connection of the HDMI cable that triggers the ADV7511 interrupt signal. In conclusion, the input values are always registered and the status of the interrupt signals can be read from the NiosII controller.

Chapter 10

Embedded Software Architecture

In the previous chapter all the cores integrated in the system, needed to achieve a functional prototype with the desired features, have been introduced and deeply explained.
All of these cores interact in one way or another with the ADV7511 HDMI transmitter and the control of all of them is offloaded to the NiosII controller.

This chapter explaines the development phases, organized from low-level to higher levels of hardware abstraction, followed to build functional embedded software for the HDMI generator board. It is important the distinction made between the platform organization and architecture, which is very important to remark the scalability of this design to future upgrades, to the software modules developed exclusively for the application of HDMI generation.

The organization of the chapter from jobs with low-level of hardware abstraction, like device drivers development, to jobs with a higher-level of hardware abstraction, like platform remote control architecture and task interaction, is made to illustrate the build up of the software in different layers.

10.1 NiosII embedded software architecture selection

In the *Annex Section B.4: SW Design Flow,* the different options the software designer has to start a development for a NiosII platform have been introduced, so this part is not explained at detail in this chapter.

Basically the HDMI generator board is the typical case of a high-complexity system with a lot of components involved, and a complex system architecture that has been developed in the previous chapters. In this situation the straightforward decision has been to use the MicroC/OS-II (uC/OS-II) real time operating system (RTOS) port for NiosII microprocessors, which Altera distributes with the NiosII Embedded Development Suite (EDS).

The uC/OS-II kernel operates on top of the hardware abstraction layer (HAL) board support package (BSP) for the Nios II processor. Because of this architecture, uC /OS-II development for the Nios II processor has all the advantages detailed in *Annex Section B.4.2: Embedded software architecture for NiosII Processor. Software layers introduction* when a RTOS is used in front of a classic super-loop application. The features that a RTOS provides are not included here, to not repeat the same concepts.

Combined also with useful middleware software like the Lightweight Internet Protocol (lwIP) software component, the ability to build networked embedded systems applications in a quick time increases dramatically, as the software designer can start directly to organize the application software, without the necessity to deal with the complex underlying hardware of the NiosII system, like the Ethernet MAC/PHY layers or other hardware peripherals. Figure 10.1 shows the architectural layers of the NiosII embedded software that is going to be developed.

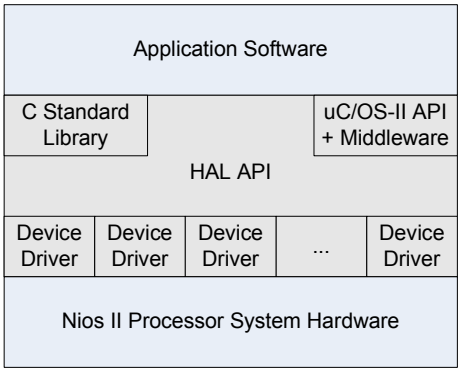


Figure 10.1. HW ⟷ HAL + Device Drivers + RTOS + Middleware ⟷ Application SW

10.2 Creation and configuration of BSP library for target system

As it is detailed in the *Annex B: Embedded System Design Methodology,* there is a tight integration between all the Altera development tools that allow the designer to cross the bridge between hardware and software design flow easily. When the NiosII system is generated with SOPC Builder, the tool creates a special SOPC Information filet that is used in the NiosII EDS to generate a system library, also called Board Support Package (BSP), specific to that SOPC Builder system. This integration allows to reliably go back and forth between the software and hardware projects.

In NiosII EDS, the application project, and the system library projects are physically separated in the environment interface, in order to distinguish which source code part does not need to be modified because it has been created automatically for the specific NiosII system designed with the SOPC Builder tool. With the SOPC information file the software design can start, and the first step is to configure the BSP project to our necessities. The dialog to make it is shown in Figure 10.2.

The only remarkable points of the BSP configuration are that uC/OS-II kernel and the lwIP software component has been selected, and configured with the default options. Some HAL settings have also been configured, to run the software from the DDR-SDRAM, and use the debug JTAG UART as the standard input and standar output, it means HAL functions like printf will use the JTAG UART, visible from the NiosII EDS environment while debugging the software using the JTAG connection.

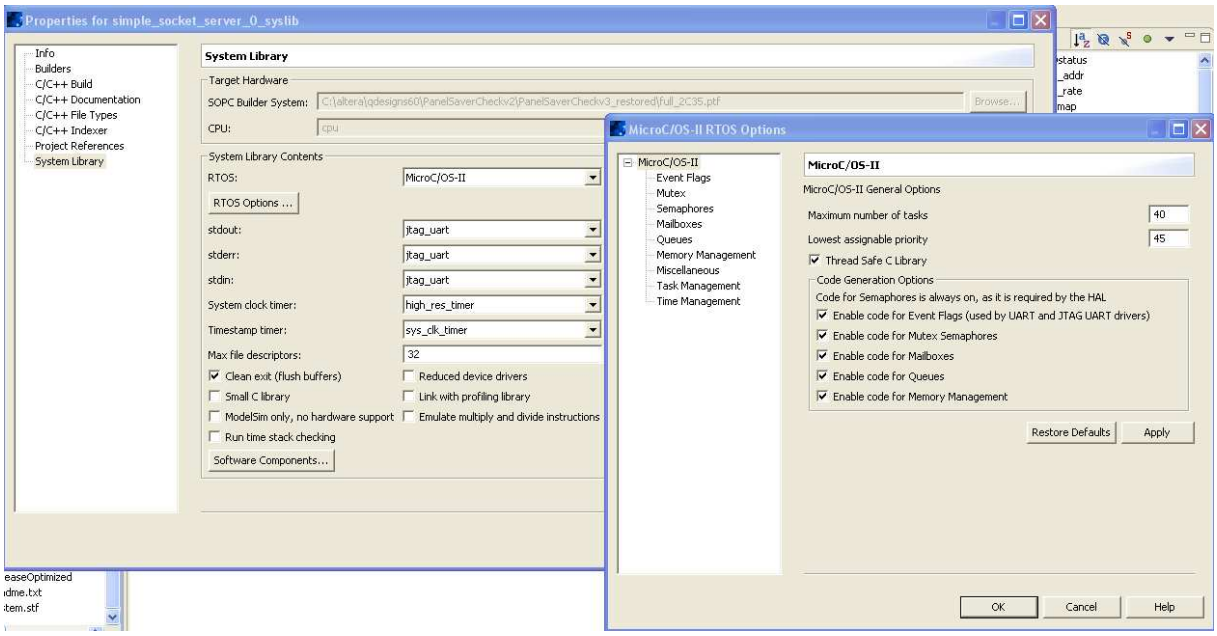


Figure 10.2. System library configuration.

The device drivers for all the cores integrated in the SOPC Builder system has also been created during the BSP configuration. For example, that means there are methods that can be used from the application software to read/write in the external Flash, to allocate memory in each of the memory devices, or to use the two timers integrated in the system..

But all these device drivers have been automatically created only for components integrated in SOPC Builder, and as it have been seen, the most important cores for the correct performance of the system has been integrated outside of the main system in the shared Wishbone bus. The next section will explain how this issue is solved.

As a summary, most of the HAL BSP settings, and the settings configured in the SOPC builder when the component was integrated in the system, are reflected in the system.h file, which provides a a complete software description of the NiosII system hardware. Especially useful for the programmer are some details about each peripheral in the system like:

- Memory map base address and address span.
- A symbolic peripheral name.
- Interrupt request (IRQ)
- Hardware configuration of the peripheral.

The system.h file is automatically generated and it must not be edited manually.

10.3 Driver development for cores integrated outside of SOPC Builder in the Wishbone subsystem

10.3.1 Writing device drivers. Hardware direct access

In the *Chapter 7.2: System interface with external IP cores* and *Chapter 8.1: Peripheral subsystem integration with NiosII system*, it have been explained the strategy to access from the main controller to cores not directly integrated to the on-chip Avalon bus, from both point of views.

The registers of these external cores has been mapped into the NiosII memory space by using the bus1 component that connects the internal Avalon bus to the shared Wishbone bus. The HAL provides C-language macros IORD and IOWR that bypass the data cache to access directly to the last value in the core register. Table 10.1 shows the available macros.

Macro	Use
IORD(BASE, REG_OFFSET)	Read the value of the register at offset REG_OFFSET in a device with base address BASE.
IOWR(BASE, REG_OFFSET, DATA)	Write the value DATA to the register at offset REG_OFFSET in a device with base address BASE.

Table 10.1. HAL Direct I/O Macros

The definitions in system.h file specify the peripheral name, base address location, and address span. In Figure 10.3 there is the description of the bus1 component in the system.h file.

```
/*
 * bus1 configuration
 *
 */

#define BUS1_NAME "/dev/bus1"
#define BUS1_TYPE "avaloninterface"
#define BUS1_BASE 0x01211000
#define BUS1_SPAN 256
#define BUS1_IRQ 9
#define BUS1_HDL_PARAMETERS ""
```

Figure 10.3. Software description of bus1 component integrated in SOPC builder

To protect against coherency issues, the access to all external cores is with their system.h name and base address symbols, instead of a hard-coded address, to guarantee successful core register access even if the core's addressable location changes for a hardware change. So, to read a register of the I^2C Master core at address 0x00 inside the bus1 memory space, the macro must be IORD(BUS1_BASE, 0x00). All the device drivers use this direct memory access macros to access the respective core registers.

10.3.2 Interrupt handling

In the *Chapter 9.5: HW interrupts & reset control* it has been introduced the necessity to handle some asynchronous interrupt requests (IRQ) from the NiosII microcontroller. Basically there are two sources of interruptions:

- I^2C Master Core, which generates an interrupt signal when a one byte transfer is completed.
- ADV7511 HDMI transmitter, which can generate an interrupt signal to notify the controller of several state changes, like HPD detection, etc.

This interrupt sources can be very useful in order to organize the main controller software but there are different ways to detect and handle them from the NiosII.

- **ISR**. First option is to connect the IRQ directly to the NiosII, and register a user-defined interrupt service routine (ISR) using the HAL functions. With this method, from software point of view, when the interrupt is enabled, the uC/OS-II kernel makes

a context switch from the code in execution to the ISR, and when the interrupt processing ends, the normal execution resumes.

- **Polling**. Second option is to register hardware interrupts in another core directly connected to Wishbone bus. With this method, from software point of view, a low priority task has to be constantly polling the register to notice a change when the interrupt is enabled.

As it is exposed in the *Chapter 9.5*, the decision must be taken from the response time point of view, and the I^2C Master core has been selected to test the two approaches, as it is the most sensible one in case the response time is not good enough. Basically the interrupt signal is asserted when a one byte transfer is completed, it means that in a continuous read or write transfer of more than one byte, the interruption has to be quickly served in a much faster time than the I^2C SCL clock cycle time in order to do not notice it.

Two type of measurement have been made to select the approach with better performance. Firstly using the SignalTap tool, that it have been already used to show the bus read/write transfers. Basically the idea is to calculate the number of system clock cycles needed to configure again all the I^2C Master core registers after the NiosII interrupt port is asserted. The two different approaches have been tested, and the results can be appreciated in the Figure 10.4 and Figure 10.5.

Figure 10.4. SignalTap captures showing time needed to clear the interrupt and idle time until core is configured with next data to send for ISR method

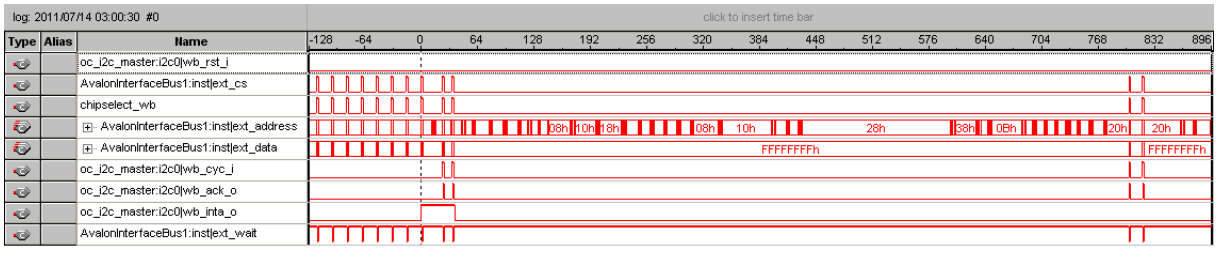


Figure 10.5. SignalTap capture showing time needed to clear the interrupt and idle time until core is configured with next data to send for polling method

From the bus signals capture it is clear to conclude that the polling method has a better response time, as it is quicker than the ISR method serving and clearing the interrupt signal. In fact two captures have been needed to appreciate the time in the ISR case because the 1k samples post trigger are not enough to capture the moment when registers are loaded again with next data to be sent. The cycles measured with SignalTap tool after the interrupt signal rising edge are summarized in Table 10.2.

Interrupt Handling Method	Time to clear IRQ (cycles)	Time until new data is configure to be sent (cycles)
ISR	38 cycles	820 cycles (approx 19 µs)
Polling	280 cycles	1100 cycles (approx 26 µs)

Table 10.2. Summary of response times measured for both methods

Eventually, in order to confirm the results of the first measurement, an I^2C read transfer of several bytes have been captured using the oscilloscope to know if the interval between two one byte transfers is appreciable or not in each of the two options. The results are shown in Figure 10.6 and Figure 10.7.

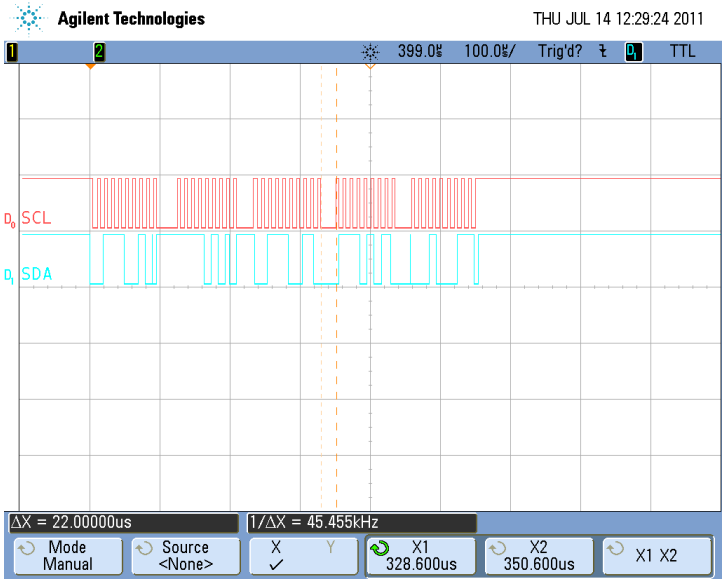


Figure 10.6. Inefficiency idle interval of 22µs with ISR method. I^2C bus speed at 100kHz

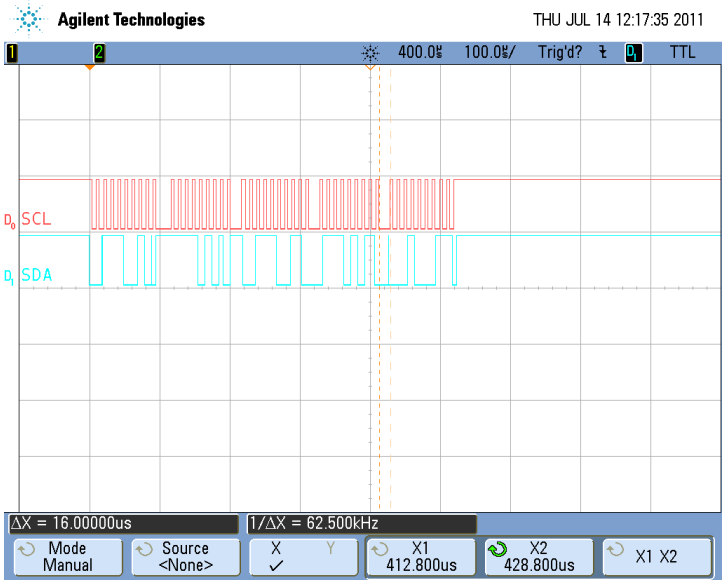


Figure 10.7. Inefficiency idle interval of 16µs with polling method. I²C bus speed at 100kHz

Finally, it is decided according the results, to use the polling method, so the interrupts are managed with a controller low priority task, which is constantly polling the Interrupts register. Particularly, in the I²C transfers, the main task is suspended while waiting for the interrupt controller task to resume the normal operation after the interrupt is processed and cleared. The implementation is made with a semaphore that blocks the main execution task until the controller task releases the resource to continue normal operation as quick as possible. It is important to remark that task management and semaphore management are both provided for uC/OS-II kernel.

Although the polling method offers a response time improvement in front of the ISR method, the response time is considered still too high and it has a room to improve, as basically the biggest loss time is during the time to free the resource and continue the main task execution that is suspended while waiting for the interrupt.

10.3.3 I²C driver development

The I²C bus to the ADV7511 is essential to configure the HDMI transmitter and as it have been explained in *Chapter 9.1: I2C Master*, an OpenCores core has been integrated in the Wishbone bus to act as the I²C master to configure the ADV7511, that acts as a slave I²C device. In *Annex F.1: I2C Master Core* the registers memory map used to control the core is included as a reference to understand the driver development.
In this section there is an outline of the main driver functions.

- `INT8U` ***`i2c_enable_core`***`(INT8U i2c_selected)`
- `INT8U` ***`i2c_disable_core`***`(INT8U i2c_selected)`
- `INT8U` ***`i2c_init_core`***`(INT8U i2c_selected, INT32U scl_clk)`

These functions are used to enable/disable the core normal operation and to configure the core at a specific I²C bus speed. The parameter `scl_clk` represents the clock frequency in Hz, so driver is in charge to write the adequate value in PRER register to get this bus speed taking into account that the core runs at the system clock frequency of 42.5MHz. The relationship between PRER, system clock and I²C bus clock can be found in *Annex F.1: I2C Master Core*

- `INT8U ***i2c_write_single_byte_isr***(INT8U i2c_selected, INT8U command_register, INT8U data, char *message)`
- `INT8U ***i2c_read_single_byte_isr***(INT8U i2c_selected, INT8U command_register, INT8U *data, char *message)`

These low-level functions are used to configure the core to send one byte of data to the slave and to receive one byte from the slave. These functions are not directly used from application software.
The parameter *i2c_selected* indicates the master core number to address, as the driver can be used also if several master cores are instantiated, although that is not the case in the HDMI generator board.
The *command_register* parameter is the value to configure the CR register, which indicates if a start or stop condition has to be generated before or after the byte is transmitted, if the byte has to be terminated with read/write bit in case the device address is transmitted, etc.
In the write function the driver internally configures the TXR register with *data* value, and the CR register with a proper *command_register* value and the WR bit flag enabled
In the read function the driver only internally configures the CR register with a proper *command_register* value and the RD bit flag enabled

In both functions, after configuring the registers, the driver suspends the execution and waits for the interrupt received for the low priority controller task. This task signals a semaphore to resume the normal function execution and clears the interrupt condition by writing the IACK bit of the CR register. If the execution is not resumed after a fixed timeout of 1500ms, the function returns with an error, as the bus is possibly stuck, it means SCL is always 0, so the I^2C Master Core cannot get access to the bus and needs to be reset, maybe because of a physical error or because in a multi-master environment another transfer is in progress. However that second option is impossible in the HDMI generator board context with only one master in the I^2C bus. In case of a write transfer, the SR register is read to decide if the slave has acknowledged the data sent by the master, and in case of a read transfer, data from slave is read from RXR register and presented to the high- level function of the driver. Both functions return a number different from 0 in case of an error.

- `INT8U ***i2c_send_individual_bytes***(INT8U i2c_selected, INT8U i2c_address, INT32U subaddr_ini, INT8U bytes_subaddr, INT32U bytes_to_send, INT8U *tx, char *message)`
- `INT8U ***i2c_read_individual_bytes***(INT8U i2c_selected, INT8U i2c_address, INT32U subaddr_ini, INT8U bytes_subaddr, INT32U bytes_to_read, INT8U *rx, char *message)`

These functions are used in the application software to write or read several bytes of data in a complete I^2C transfer, and use the low-level functions introduced just before.
In both cases the transfer starts sending the device address with the write flags, followed with the slave memory subaddress, which can have a variable number of bytes from 0 to 4.
After addressing to a specific memory position, in case of a write transfer, all bytes of data are transmitted, with the last byte terminated with the stop condition. All the one-byte write transfers use the function ***i2c_write_single_byte_isr*** with the appropriate *command_register* values in every phase of the multi-byte write transfer.
After addressing to a specific memory position, in case of a read transfer, a repeated start condition is generated sending the device address again with the read flag enabled, later data

is read constantly and the master core acknowledges the transfer. All the one-byte read transfers use the function ***i2c_read_single_byte_isr*** with the appropriate *command_register* values in every phase of the multi-byte read transfer.

- INT8U ***i2c_write_masked_reg_adv7511***(INT32U subaddr_reg, INT32U bytes_to_send, INT8U *tx, INT8U *mask_tx, char *message)
- INT8U ***i2c_write_direct_reg_adv7511***(INT32U subaddr_reg, INT32U bytes_to_send, INT8U *tx, char *message)
- INT8U ***i2c_read_direct_reg_adv7511***(INT32U subaddr_reg, INT32U bytes_to_read, INT8U *rx, char *message)

Finally a wrapper functions to directly access the registers of the ADV7511 have also been developed. As the ADV7511 memory map is fragmented at bit-level, a helper function to access registers with masks ***i2c_write_masked_reg_adv7511***, has been developed. It encapsulates a read access with a write access to modify only the desired bits of the register and left the other bits unchanged.

It is important to remark that the driver suspends the execution during the interval where data is sent or received. As the I^2C bus is a low-speed bus, this behavior allows the uC/OS-II operating system to execute other tasks with a higher priority than the controller task that is polling the interrupts register to know if the one-byte transfer has been completed. Anyway the driver can also be used in blocking mode, in this case the execution is not suspended and the driver constantly queries the interrupts register. That can be useful to assure that two different tasks do not take access to the shared resource, the I^2C bus, at the same time.

10.3.4 Video driver development

The Video Generation core is essential to send video data to the HDMI transmitter as it has been explained in *Chapter 9.2: Video generation*. In *Annex F.2: Video Generation Core,* the registers memory map used to control the core is included as a reference to understand the driver development.
In this section there is an outline of the main driver functions, but in case of the video generation driver, these functions are only hiding the direct hardware access with IOWR and IORD macros and the meaning of each of the core low-level registers. Although in this project is not so important, this style is useful when software development is absolutely separated from hardware development, in order to offer a friendly API to the application software developer.

- void ***video_change_pixclk***(char pix_clk[16])
- void ***video_timings_change***(INT16U Hactive, INT16U Hfront, INT16U Hsync, INT16U Hback, INT16U Vactive, INT16U Vfront, INT16U Vsync, INT16U Vback, INT8U hparams_update_skip, INT8U vparams_update_skip)
- void ***video_controlreg_change***(INT8U Hpol, INT8U Vpol, INT8U num_bits, char scan_type[64], INT8U hpol_update_skip, INT8U vpol_update_skip, INT8U scan_type_update_skip, INT8U num_bits_update_skip)
- void ***video_pattern_change***(INT8U pattern)

The function ***video_change_pixclk*** is used to change the pixel clock used to generate the timings. The function ***video_timings_change*** is used to change the active/blanking pixels/lines that define the video timing as explained in *Chapter 9.2: Video generation.* The

function ***video_controlreg_change*** is used to change the core control register values like the progressive/interlaced flag as well as the sync pulse polarity. Finally the function ***video_pattern_change*** is used to switch between the diifferent video patterns available. As the core can be partially updated, it means to change only a subset of the registers, all the functions of the drivers also support partial updates with the skip option flags.

10.3.5 I²S driver development

The I2S Generation core is essential to send audio to the HDMI transmitter as it has been explained in *Chapter 9.4: I2S generation.*In this chapter the device driver development is explained. In *Annex F.4: I2S Generation Core,* the registers memory map used to control the core is included as a reference to understand the driver development.
In this section there is an outline of the main driver functions created to offer a friendly API to the application software developer.

- `void` ***audio_configure_mute***`(char i2s_mute[64])`
- `void` ***audio_configure_jump_count_left***`(INT32U jump_count_left)`
- `void` ***audio_configure_jump_count_right***`(INT32U jump_count_right)`
- `void` ***audio_configure_sampling_frequency***`(char freq_str[64])`

The function ***audio_configure_mute*** is used to configure the audio mute independently for left and right channels. The function ***audio_configure_jump_count_left*** is used to change the left channel frequency by changing the ROM jump size as it has been explained in *Chapter 9.4: I2S generation.* The function ***audio_configure_jump_count_right*** has the same behaviour but for the right channel frequency. Finally the function ***audio_configure_sampling_frequency*** is used to change the core sampling frequency, that also affect the frequency output of the audio tones.

10.4 Application software architecture design

10.4.1 Importance of planned software architecture in RTOS-based systems

As it has been explained the application software running on the HDMI generator board is based on uC/OS-II RTOS. The uC/OS-II is a multitasking preemptive kernel; it means that the highest priority task ready to run is always given control of the cpu. For example if a low priority task is suspended (preempted) because of an ISR, later when the ISR completes, the higher priority task is resumed, so it is possible that the task interrupted do not take control of the cpu. The main advantage of the preemptive kernels is the system responsiveness, it means that execution of a high priority task can be specifically planned and forced, while in non-preemptive kernels it is possible that a high-priority task ready to run might have to wait for a long-time while a low-priority task is taking the cpu. Although it is a better option in general, the drawback of preemptive kernels is that shared data has to be correctly protected with semaphores, because tasks never know when they can lose the cpu, maybe in the middle of a critical operation, and that priority assignment must be planned with care as it is not trivial. To clarify the notation used, in uC/OS-II the priorities are reverse-ordered, it means the task with the lower priority number is the higher priority task.

It is important to remark, that the explanation in this chapter is centered in the software behavior more than in how the tasks are created and implemented in the uC/OS-II context.

For more details about how has been developed in code using HAL API, uC/OS-II services API and lwIP functions, some useful references are provided. *[49][50][51]*

10.4.2 HDMI Generator board software architecture details

The application software is build around the remote control ethernet interface based on a simple socket server implementation. Exactly like in the system architecture design, the small part regarding lwIP TCP/IP stack initialization is based on the core of one of the several examples to use the Nios 2C35 kit. Anyway, only the core of lwIP and uC/OS-II initialization is kept as is, and the rest of the socket server implementation has been all modified to adapt to the planned embedded system. All the rest of application software is created from scratch.

The Table 3.1 lists all the tasks created in some moment for the application and give a brief introduction of which is the purpose of each task.

Task	Priority	Description
SSSInitialTask()	1	Initializes the OS data structures. The task deletes itself after initialization completes.
NETUTILSDHCPTimeoutTask()	2	Sets a static IP address after DHCP timeout expires. The task deletes itself after initialization completes.
LWIPRxEtherTask()	3	This lwIP task is responsible to collect packets from the Ethernet device and pass them to the TCP/IP stack. The priority has to be high to avoid dropping any packets.
LWIPTCPIPTask()	6	This lwIP task is responsible to process packets from the other lwIP task, and make them available to the application software through the sockets API calls. In order to maximize the TCP/IP packet throughput rate, the priority of this task should be higher than application software tasks that use socket API calls.
SSSSimpleSocketServerTask()	9	This is the most important task in the system, because it creates a socket to serve any connections, binds to the socket and listens for connection requests from a client. Every time an incoming connection arrives it is served by this task that processes the command information and enables other tasks if it is required by the processed command. This task has the higher priority of all the application software tasks.
SSSSocketServerSendTask()	10	This task is responsible of sending the command response back to the opened socket when the command processing and execution is finished.
ADV7511InitialTask()	11	This task is responsible to enable the HDMI transmitter interrupts and start the interrupts controller task after the initialization of the socket server has been completed.
ADV7511ControlTask()	12	This task parses the commands to control the HDMI generation operation and execute the desired configuration changes depending on the meaning of each command.
ADV7511InterruptsProcessing()	13	This task is created just after detecting an ADV7511 interrupt in order to process the interrupt and change the ADV7511 operation mode if required.
ADV7511GetInformation()	14	This task is created to cyclically compile information about ADV7511 status and send it by the open socket.
I2CWriteReadTaskI2C1()	20	This task parses the command to control the I2C bus, and is used if direct access to I2C bus from the remote control

		interface is required.
I2CWriteReadTaskController()	21	This task is created to monitor the I2C Master Core interrupt line.
ADV7511InterruptsDetection()	22	This task is created to monitor the ADV7511 interrupt line.

Table 10.3. uC/OS-II tasks for the HDMI generator board

Each of these system tasks has a priority assigned, and Figure 10.8 shows the initialization process with all the tasks involved ordered by the priority assignment.
The process of system initialization is:

1. The lwIP stack is initialized. The low-level lwIP TCP/IP task LWIPTCPIPTask() is kept alive in suspended mode, as is in charge of collecting packets from ethernet device.
2. The OS data structures and other tasks are created, and the OS enters multitasking. This task SSSInitialTask() with the higher priority kills itself after execution.
3. The low-level task NETUTILSDHCPTimeoutTask() to set a MAC Address and initialize the Ethernet device is created. This task is also kept alive in suspended mode, and in charge of processing TCP/IP packets to serve it to upper layers application tasks.
4. The DHCP timeout task is initialized and polls for a DHCP server every 100ms during a fixed time of 1500ms. It means if DHCP is not found task is suspended during 100ms, and other tasks with less priority get the cpu.
5. The Simple Socket Server SSSSimpleSocketServerTask() task, SSS task from now on, is created to listen for a socket connection and handle the connection. This is the main task of the system as is responsible to accept connections and process all the messages received from the client. Once setup, SSS task perpetually waits for incoming data to either the listening socket, or (if a connection is active), the SSS data socket. When data arrives, the appropriate routine is called to either accept/reject a connection request, or process incoming data.
6. After the DHCP is found or timeout expires, an IP address is assigned, and the IP semaphore is released, so the socket server is ready to accept connections.

After the system is initialized, the timing diagram also shows the interaction between tasks in a typical connection process. The lwIP ISR interrupts the tasks currently in execution, to resume the execution in the two lwIP tasks that always have the higher priority. It means that the system is responsive to any connection, because any task executed is suspended to process Ethernet traffic by the ISR, and the normal execution will always resume with the lwIP tasks and the Simple Socket Server task, because these tasks have been intentionally assigned with a priority higher than any other application task.

7. From now on, the low-level lwIP tasks executed before SSS task will not be mentioned, although they are part of the process of receiving ethernet traffic.
8. If a client is trying to establish connection and there is no other established connection, the SSS task accepts the connection, and creates the SSSSocketServerSendTask() task, Send task from now on, used to communicate the command execution result to the client. This task also has a higher priority than any other task in the system, and after the connection is established, the task is constantly monitoring an OS message queue where other tasks can push messages to be sent back to the client. That is a common method for task intercommunication when the system is based in a RTOS that supports that service.

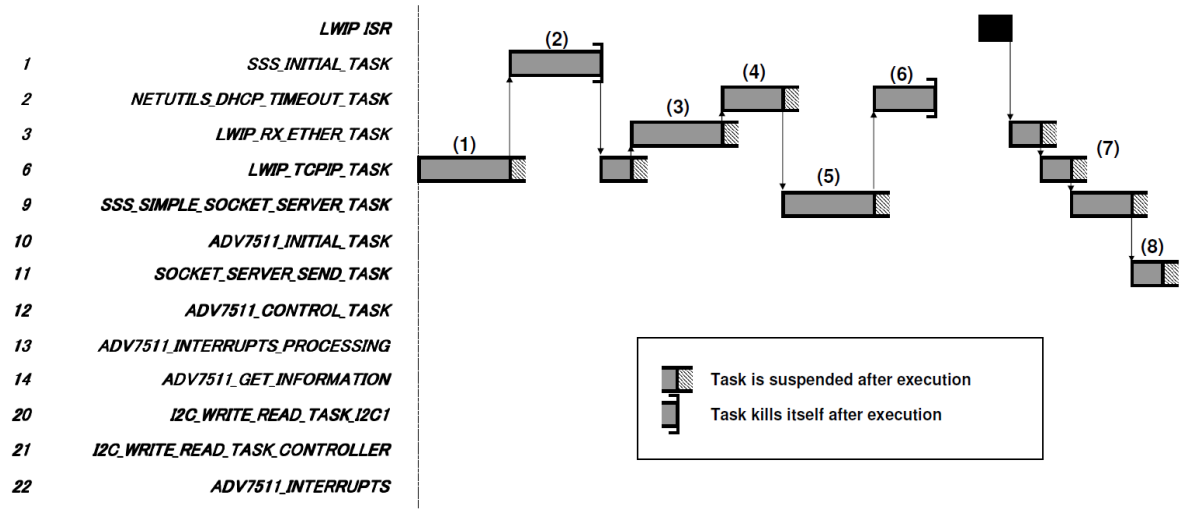


Figure 10.8. Tasks interaction timeline at system initialization ordered by priority

10.4.3 Simple Socket Server command processing

Once the socket connection is established, the simple socket server (SSS) task is constantly receiving incoming data until a line feed (LF) character is received. In a text-mode socket client like Microsoft Telnet client that equals to push the ENTER key. The entire reception data buffer until that moment is interpreted as a command and the SSS task tries to execute it. The command parser splits the command in two tokens using the space separator, and the first token is taken as the command name. The SSS task checks in the list of commands to see if the token matches one of the accepted commands, and in the positive case it executes the task associated to that command. The second token is passed to the task as it contains all the command arguments that define the task execution.

Annex G: HDMI Generator control commands list contains the full reference of commands accepted by the HDMI generator board.

Figure 10.9 shows an example of the SSS command processing for an accepted I^2C command with all the interaction between tasks involved:

1. SSS task receives the command and separates the command name from the arguments in its simple parser. If command is accepted in the HDMI generator board, the SSS task executes the corresponding task to serve the command, in the example the I2C1 task for direct I^2C bus access. The arguments of the command are passed to the task for further processing.
2. I2C1 task, I2CWriteReadTaskI2C1(), created by the SSS task processes the arguments in its own advanced parser. All the arguments are separated by space character separators and passed in the format –key <value 1> ..<value n>, although sometimes the first token has a special meaning defining task mode. For more information refer to *Annex G: HDMI Generator control commands list*
   The task also creates the I2C controller task, I2CWriteReadTaskController(), in charge to monitor the interrupts and uses the appropriate I2C driver function calls depending on the arguments processed.
3. The I2C controller task is in charge to monitor the interrupts and resume the main I2C1 task, when the corresponding interrupt signaling the completion of a transfer is received. This process is repeated for any one-byte transfer to the I^2C bus.

4. Finally when I2C transfer is completed and I^{2}C driver call returns, the I2C1 task push a message in the shared message queue with the Simple Socket Server Send task, to indicate the result of the command execution and deletes itself.
5. The Send task sends the command response to the connected client indicating if the command has been successfully processed or not.
6. The low-priority I2C controller task deletes itself when its execution is resumed, as the transfer has already been completed, and the I^{2}C Master Core interrupt does not need to be monitored regularly.

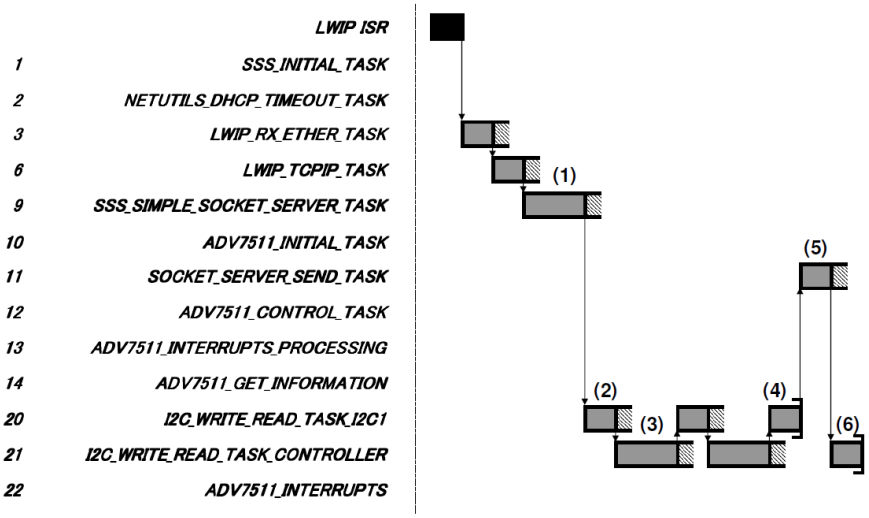


Figure 10.9. Tasks interaction timeline in command processing ordered by priority

As a conclusion, the way to interact externally with the HDMI generator board has been introduced, and the general idea is that these application tasks, that can last longer if the command executed does not return immediately, can always be interrupted if new data is received by the SSS task, because they are organized with low priority assignments.

10.5 ADV7511 HDMI generation control

10.5.1 ADV7511 software driver functions development

The following is a list of all the functions implemented to control the ADV7511 HDMI transmitter and a description of the operation, parameters and return values of the functions. All these functions internally use I^{2}C driver calls to configure the different blocks of the HDMI transmitter, and again the main purpose is to hide the complexity of the huge ADV7511 memory map from the software developer. Also any improvement in the driver solving some issue, is quickly deployed in all the application software that use the common driver interface.
As the ADV7511 datasheet and programming guides are under a NDA, the driver details are not exposed, and only a behavioral description is provided.

ADV7511 initialization functions

```
INT8U adv7511_check_init(char *message);
```

This function checks that the I2C bus to the HDMI transmitter is ready. Depending on the power-up state of the PD pin, the transmitter can select two different I2C addresses by default, so this function only discovers the correct I^2C address to make all the transfers.

```
INT8U adv7511_config_initial_regs(char *message, INT8U i2c_map_reload);
```

This function is used to configure the HPD and RxSense detection in the ADV7511. Both methods are used to detect if the HDMI sink is connected to the transmitter. RxSense refers to the detection of TMDS clock line pull-ups in the HDMI sink. One strong reason to detect the RxSense is to delay the chip power up until the HDMI receiver is actually ready to receive signals. Typically in the sinks the HPD line is directly connected tied to the HDMI +5V from the source through a series resistor. In that case, the HPD signal is detected regardless of whether the sink is powered on or not, so is best to wait also to the RxSense detection.

```
INT8U adv7511_mask_interrupts(char *message);
```

This function enables the interrupts for the following events: HPD and RxSense detection, EDID ready, and DDC/HDCP error in the DDC channel between ADV7511 and the HDMI sink. In case of HPD and RxSense, the interrupt is triggered in any transition, so the state of the associated registers has to be queried.

ADV7511 power up/down state functions

```
INT8U adv7511_main_power_up(char *message);
```

This function powers up the ADV7511 by setting high the Power Down register bit, but the state is not kept if the HDMI sink is not detected, it means RxSense and HDP are in low state. After powering up the device, some fixed values registers must be set for proper operation following datasheet advice.

```
INT8U adv7511_main_power_down(char *message);
```

This function powers down the ADV7511 by setting low the Power Down register bit.

ADV7511 main configuration functions

```
INT8U adv7511_main_operation_setup(char *message);
```

This function configures the ADV7511 operation mode between HDMI or DVI options, and disables some features of the ADV7511 that are not used in this application.

- HDCP disabled.
- CEC block disabled and powered down.

```
INT8U adv7511_main_video_setup(char *message);
```

This function configures the video setup of the ADV7511. First of all the registers used to gather information about the video input signal, are configured according the video generation core output signal:

- RGB 4:4:4 video input signal format.
- Appropriate color depth depending on video generation core configuration.

Later, the input signal information registers used to detect the Video Mode between all the CEA-861-E video formats are configured:

- Aspect Ratio.
- Pixel repetition is disabled as VICs with pixel repetition are not used.

Some features of the video block are disabled because they are not used in this application:

- Internal color space converter disabled.
- Sync generation disabled.

Finally, only in HDMI mode as DVI mode has no control packets, the General Control and AVI InfoFrame packets must be enabled, and the information needs to be configured.

- GC packet enabled and color depth information configured in the packet.
- AVI InfoFrame packet enabled. Aspect ratio and color depth information configured in the packet.

```
INT8U adv7511_main_audio_setup(char *message);
```

This function configures the audio setup of the ADV7511 when the transmitter is configured in HDMI mode, as in DVI mode cannot send audio packets.
First of all the registers used to gather information about the audio input signal are configured according the I^2S generation core output signal:

- I^2S capture block enable.
- Sampling frequency configured according the I^2S generation core configuration.
- Only first of the four I^2S Data lines enabled. 2:0 audio channels configuration.

One of the most important parts to configure in order to assure that audio is correctly received by the HDMI sink is the audio clock regeneration. As explained in the HDMI technology introduction, audio data carried across the HDMI link does not preserve the original sample clock, which has to be regenerated in the sink from the video pixel clock. The audio regeneration model is based on the rational relationship between these two clocks. The source shall determine the fractional relationship between the video clock and the audio reference clock ($128 \times Fs$) and shall pass the numerator (N) and denominator, also called cycle time stamp (CTS) of that fraction to the sink. The relationship is shown in equation (10.1).

$$128 \times Fs = f_{TMDS\_CLK} \frac{N}{CTS} \qquad (10.1.)$$

The audio setup function disables the automatic CTS calculation, and determines the N and CTS values, depending on the sampling frequency of the input audio signal, to configure the appropriate 20-bit registers. It also enables the special Audio Clock Regeneration packet.
The audio sample packets also have to be enabled and the information about the audio data carried across the HDMI link has to be inserted in the packets.

- Channel count. Stereo 2:0
- Default packets ↔ channels mapping.
- Audio word length fixed to 16 bits.

Finally, the Audio InfoFrame packets are also enabled to configure the speakers mapping for the audio packets.

```
INT8U adv7511_av_mute(char *message, INT8U on_off);
```

This function enables/disables the AV mute request to the sink that is transmitted inside the General Control packet. This request is useful in case the audio or video format is going to change in order to make the possible momentarily artifacts in video or audio signal, not visible to the sink final user. For example in case of a TV acting as a sink, the audio and video can be muted during the format changes.

ADV7511 helper functions

```
INT8U adv7511_packet_update(char *message, INT8U on_off, INT8U
packet_to_update);
```

This function is used to momentarily fix the latest packet information during the packet information configuration. This is a useful feature of the ADV7511 transmitter, that allows to "freeze" momentarily the HDMI packets sent (`on_off=1`), in order to apply all the changes at the same time when the function "unfreezes" (`on_off=0`) the packet update. That makes nearly impossible to send across the HDMI link, packets in a temporary state, in the middle of information configuration changes. It is used every time a packet needs to be updated.

```
INT8U adv7511_mask_rx_sense_interrupt(INT8U enable_interrupt, char
*message);
```

This function is used to momentarily disable the RxSense interrupt detection when the HDMI is powered up and correctly transmitting the HDMI signal. That is made because RxSense can make some glitches when the sink makes an input change or so on, and if it is masked, the power down is only made in a HPD falling edge. That is enough to make the transition Power Up → Power Down. Anyway the interrupt is unmasked again in power down mode, as the two conditions, HPD and RxSense high, are always checked to make the transition Power Down → Power Up.

ADV7511 informational functions

```
INT8U adv7511_mask_rx_sense_interrupt(INT8U enable_interrupt, char
*message);
```

This function is created to be periodically called and gathers information about different ADV7511 status registers:

- ADV7511 power mode status.
- ADV7511 chip revision.
- Power Down pin polarity.
- HDP state.
- RxSense state.
- PLL lock status.
- HDMI/DVI mode.
- Video Identification Code (VIC) detected for the current input video signal.
- I^2S Mode detection (16-32 bits)
- …

If there is an active socket connection established, all the information gathered is sent back to the client connected to the HDMI generator board control interface.

ADV7511 interrupt handling functions

```
INT8U adv7511_read_interrupts_regs(char *message);
```

This function implements the recommended flow for processing ADV7511 interrupts. More detailed information can be found in the next section.

10.5.2 ADV7511 interrupt handling functions development

As it was explained in *Chapter 9.5: HW interrupts & reset control* the ADV7511 interrupt pin connected to the FPGA enables the possibility to implement an interrupt-driven system controller, reducing the need of I^2C monitoring traffic at defined intervals between the NiosII and the ADV7511.

In the initialization the interrupts for the following events has been enabled by calling ***adv7511_mask_interrupts*** driver function: HPD and RxSense detection, EDID ready, and DDC/HDCP error.

Figure 10.10 shows the flow for processing ADV7511 interrupts that has been implemented in the ***adv7511_read_interrupts_regs*** ADV7511 driver function.

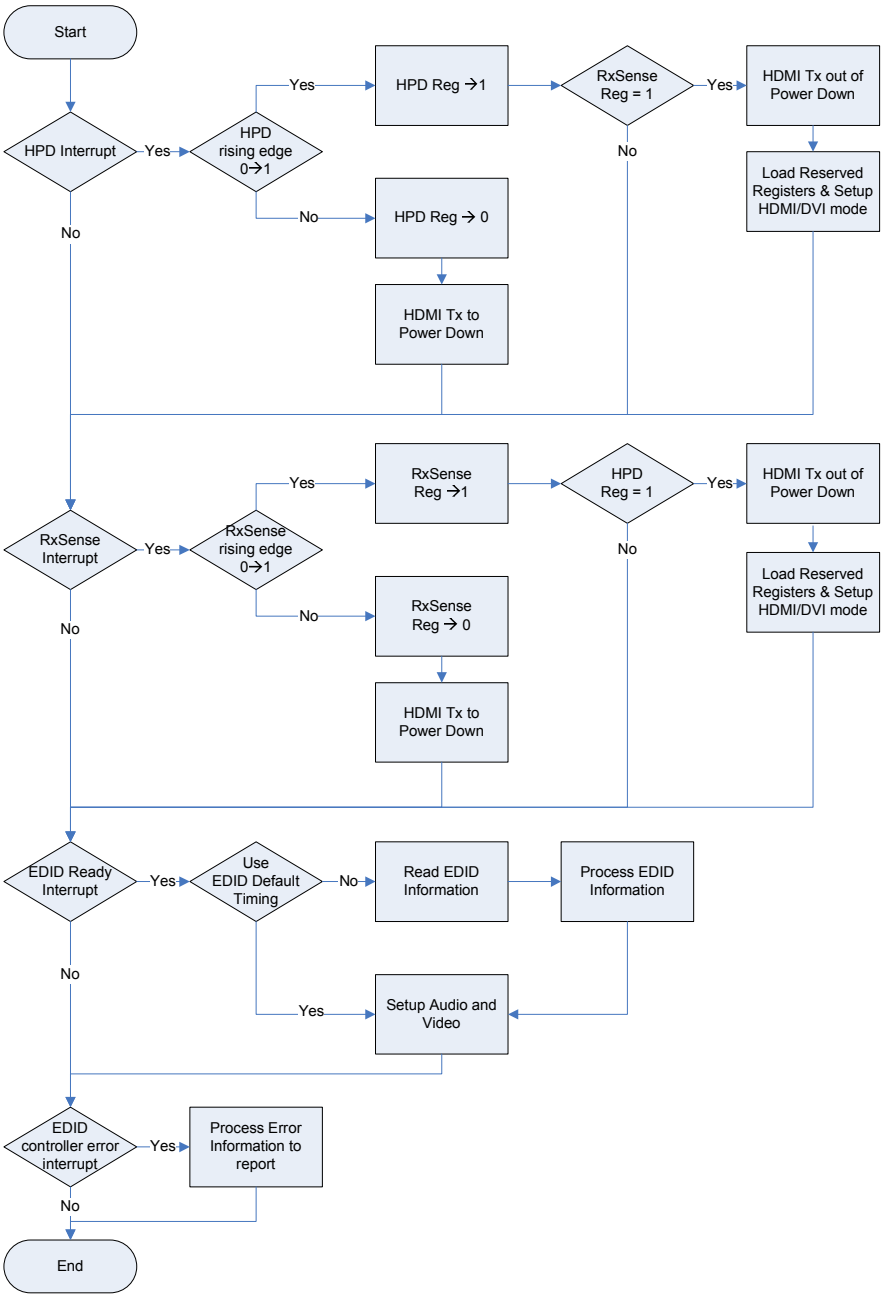


Figure 10.10. ADV7511 interrupts processing flow

In case of HPD and RxSense, the interrupt is triggered in any transition, so the state of the associated registers has to be queried in order to know if the interrupt has been enabled because of a falling or rising edge.

1. Both HPD and RxSense must be high to power up the ADV7511 and make the initial setup by calling ***adv7511_main_power_up*** and ***adv7511_main_operation_setup*** driver function calls.
2. When the ADV7511 is powered up, immediately tries to read the sink EDID because it is mandatory in the HDMI specification *[20]*. After the EDID read is completed, the EDID Ready interruption is triggered and the system can configure the audio and video setup by calling ***adv7511_main_video_setup*** and a***dv7511_main_audio_setup*** driver function calls.

Basically, this behavior means that the system can be properly initialized, but is kept in a dormant state until the interruptions are detected, processed and cleared. Only after the sink EDID has been read the HDMI link is finally activated.

10.5.3 ADV7511 and input signal cores initialization

As the main purpose of the HDMI generator board is to always generate an HDMI signal with the selected configuration parameters, a place needs to be found to initialize the HDMI related parts without necessity to make it with an external command.

Figure 10.11 shows how the ADV7511 Initial task is called from the SSS task to initialize the video and audio generation cores as well as the ADV711 HDMI transmitter.

1. As this task has a lower priority than the SSS task it is only executed when the SSS task is suspended waiting for an incoming connection.
2. The ADV7511 Initial task enables if needed, the audio and video generation cores, and also initializes the ADV7511 by calling the initialization ADV7511 driver function calls. Later this task executes the low-priority ADV7511 interrupts detection task that is constantly polling to know if the ADV7511 interruption is triggered, before killing itself. The interrupt controller task has been used again in front of an ISR because of the slow NiosII interrupt handling but at least the polling does not need to be made with I^2C bus transfers to read directly the ADV7511 registers.

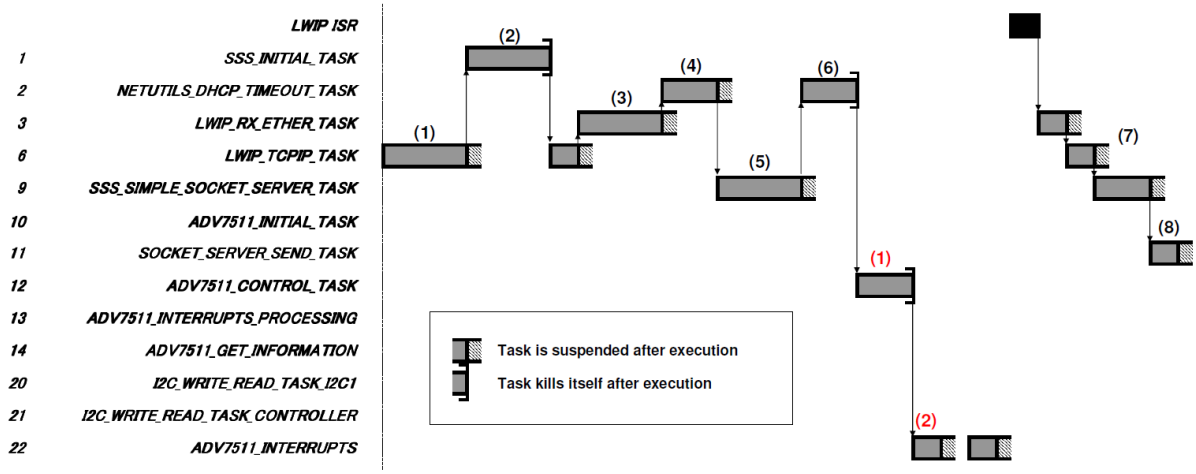


Figure 10.11. Tasks interaction timeline in ADV7511 system initialization

10.5.4 ADV7511 normal tasks interaction after initialization

This section introduces the tasks interaction when the HDMI generation is running normally and some external event interrupts the normal flow.

- The socket receives an accepted ADV7511 command to start the system when it is stopped. *Annex G: HDMI Generator control commands list* contains the reference for the HDMI generation control commands accepted by the HDMI generator board.
- An HDMI sink is connected to the HDMI generator board.

Figure 10.12 shows the interaction between tasks during these two events.

1. SSS task receives the command and separates the command name from the arguments in its simple parser. As the ADV7511 command is accepted it executes the task ADV7511 control task.

2. ADV7511 task created by the SSS task processes the arguments in its own advanced parser. The first token has an special meaning that defines if the command is used to change the video or audio generation cores configuration, or to start/stop the ADV7511 HDMI generation. For more information refer to *Annex G: HDMI Generator control commands list*
   In the example, the ADV7511 system is supposed to be stopped and the command is used to start it, so the task creates the low-priority ADV7511 interrupts detection task that is constantly polling to know if the ADV7511 interruption is triggered. In case of a video or audio command, the task would make the configuration and would kill itself after the execution.
3. The ADV7511 interrupt controller task is in charge to monitor the interrupts and executes the ADV7511 Interrupts processing task, when the corresponding interrupt signaling an event in the HDMI transmitter is received. After executing the new task, as this one has always a higher priority, the controller task always loses the execution to the newly created task to process the interrupt.
4. This ADV7511 interrupts processing task calls the ***`adv7511_read_interrupts_regs`*** ADV7511 driver function explained before in this chapter, to process the interrupt and clear it. It also executes the ADV7511 get information task that cyclically polls the transmitter to get information about the HDMI generation status.
5. The ADV7511 get information task is executed every time the configured timer interval expires and it executes the ***`adv7511_mask_rx_sense_interrupt`*** ADV7511 driver function call, which gets the information and sends it to the client using the active socket connection.
6. The low-priority ADV7511 interrupt controller task loses the execution every time the get information task is ready to execute, as this task always has a higher priority.

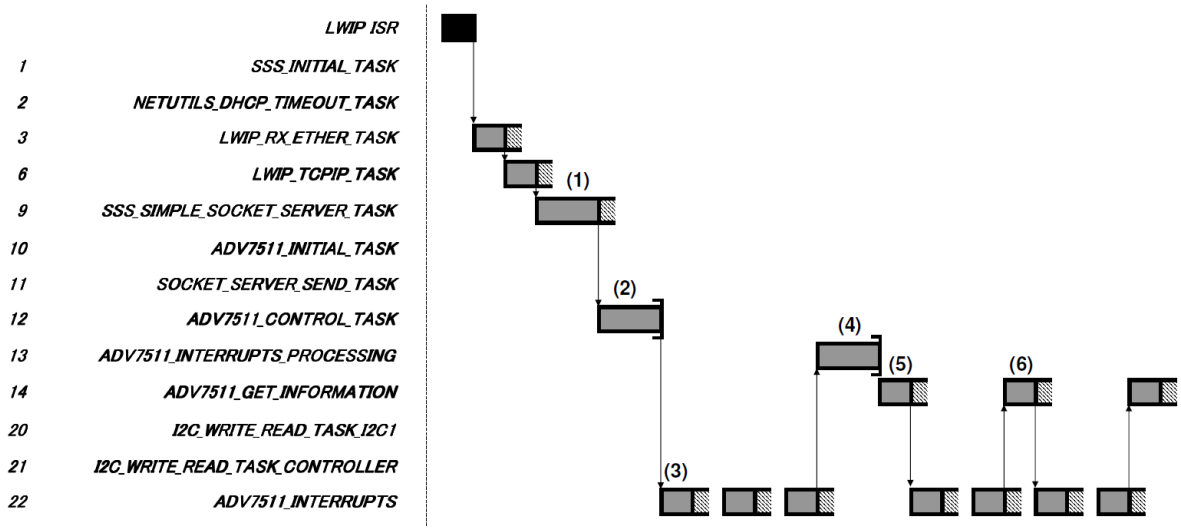


Figure 10.12. Tasks interaction timeline in ADV7511 command processing

Chapter 11

Power up, Verification & System validation

The process to mount for the first time an electronic system and make it work is typically known as the power up. Sometimes in a complex electronic system this process has to be made with care and step, it means the general process has to be broken down in phases in order to assure one phase before starting to test the next one that depends on the other. It is important to follow strictly this methodology to speed up the power up process. On the opposite, to put all the system in place and test the final functionality is not recommended, because in case of failures it is more difficult to find the real root causes that make the system fail.

In this chapter all the power up process is explained step by step, with a special focus in the methodology to make it. Anyway in complex systems with a lot of hardware/software codesign sometimes is very difficult to break down the power-up process in phases, as a preliminary software system is needed to test nearly all the hardware functionalities, but the chapter also explains an strategy to do it in successive approximations to the final functionalities.

11.1 General test equipment and requirements for verification

Specific equipment and instrumentation are required to check all the different blocks of the system in successive phases.

- Laboratory power supply to independently power up the HDMI transmitter board assuring no shorts have been made during the manual soldering process.
- Host PC with an Ethernet Network Interface Card (NIC) to control and monitor the HDMI generator board through its Ethernet interface.
- TV with Factory Mode enabled, to check one by one all the features of the HDMI generator are working correctly.
- Mixed-Signal Oscilloscope (MSO) with 2 analog channels plus 16 digital logic channels to check the prototype internal signals from master controller to ADV7511, as well as the other control signals in the HDMI connector, as well as the audio output of the TV when HDMI input is selected.

In the phases detailed below in this chapter, it will be explained further why this equipment is needed and how it is configured.

11.2 Power up & Initial system verification phases

11.2.1 HDMI transmitter daughter board mount and initial power up. Hardware verification

The first step when trying to power up and validate the complete system design is to mount the HDMI transmitter daughter card that is used as an expansion to the 2C35 development kit to develop the HDMI generator.

The design flow was linear starting from the HDMI transmitter board schematic and PCB, and there was a requisite of short time-to-prototype, so manufacturing data was quickly send to the PCB manufacturer before starting the next phase of the project. During the PCB manufacturing time, the design of the main system to power-up the board was started and quickly a concept flaw was found in the first version of the prototype sent to the PCB manufacturer.

The FPGA PLL resources were quickly found insufficient to generate the desired clocks, needed for the audio and video generation blocks. The simple solution further explained with details in *Chapter 8.2: Clock network design*, was to add a pair of external oscillators, but it was decided to not pay the extra cost to send another version to the PCB manufacturer, and the option to rework the initial prototype was chosen. The Figure 11.1 shows the aspect of the HDMI generator board after applying the clocks rework. The footprint to solder the oscillators have been manually made in the board copper pour, and the connections to the desired pins of PROTO connectors, corresponding to clock input pins of the 2C35 kit, have been made using wire-wrap.

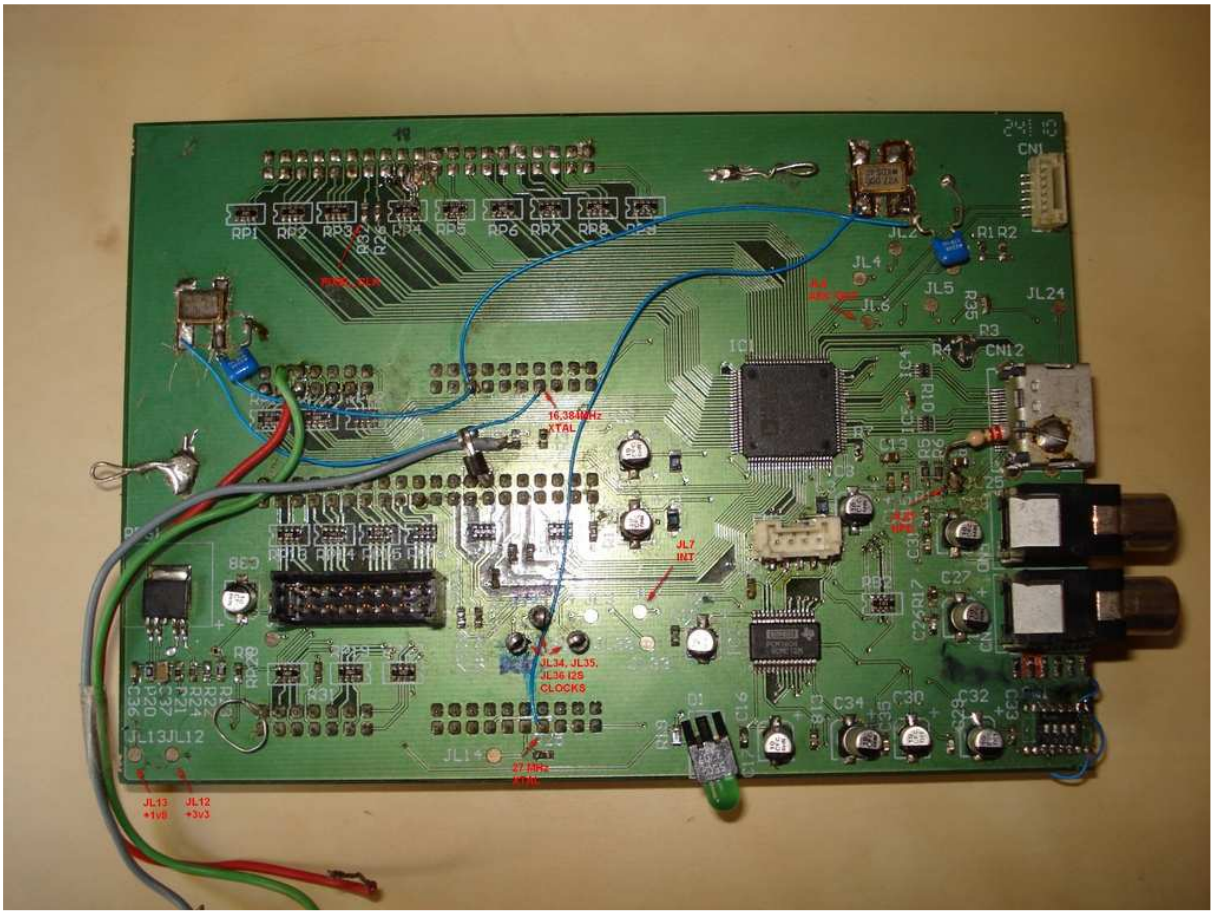


Figure 11.1. HDMI transmitter board mounted and with clocks rework applied.

After applying the rework the initial power-up is made by supplying the voltage to the two power rails, +5V and +3.3V with an external power supply instead of directly connecting the board to the 2C35 kit. The main advantage is that the laboratory power supply has a configurable current limiting function for each of the outputs. With its overcurrent protection circuit, it can be verified that the prototype mounted does not have any shorts and that the current consumption is the expected one.

After applying the voltage in the main power rails, the adjustable voltage regulator is verified in jigland JL13 to check it works as expected, and +1.8V are present in the regulator output. This check assures that no error have been made when selecting the resistor bridge values for the adjustable regulator.

The clocks rework is also verified by checking the oscillator frequency in the specific pins of the Santa Cruz expansion connectors.

The next phase consists on checking the behavior of the HDMI transmitter board assembled with the 2C35 development kit, but to make it, the hardware image must be downloaded to the 2C35 kit. It is important to assure that the correct design with the desired pin assignments is downloaded to the 2C35 platform, and that the FPGA is correctly configured, in order to avoid some pin incompatibilities that could damage the boards, for example, a dangerous situation could appear if pins that should be configured as inputs for this system in the FPGA, are wrongly configured as outputs in the 2C35 factory image.

11.2.2 HDMI generator system verification. Direct hardware verification without the control interface

The first step is to download permanently the base hardware design to the target board, the 2C35 development kit.

1. The USB Blaster download cable from the host computer is connected to the EPCS configuration connector, to download the design to the EPCS serial flash configuration device.
2. In the Programmer tab of the Quartus II software, the USB Blaster is configured and the Active Serial Programming method is selected as the programming mode to permanently download the design to the platform.
3. The appropriate hardware image with .pof extension is selected and downloaded to the board.

After a complete reset, the FPGA is configured as specified by the hardware design, and now the platform is ready to download software and run applications created for this specific configuration based around a NiosII processor. So, the next step is to download the NiosII software permanently to the flash memory of 2C35 kit:

1. The USB Blaster download cable from the host computer is connected to the FPGA JTAG connector, to download the design to the flash memory device in the 2C35 board through the flash controller configured in the FPGA after hardware image is correctly loaded.
2. With the Flash Programmer tool integrated in the NiosII Embedded Design suite, the USB Blaster is configured.
3. The Flash programmer tool generates the .elf binary image for the current software project and downloads it to the flash memory. Before downloading the image the programmer also performs a system ID and a timestamp check, to assure that the software has been developed and compiled targeting the actual hardware image configured in the target FPGA. That feature is very important during the hardware/software codesign to check the integrity between hardware and software.

Now that 2C35 development kit has been prepared for the designed system, the HDMI transmitter board can be assembled in the expansion connectors as shown in

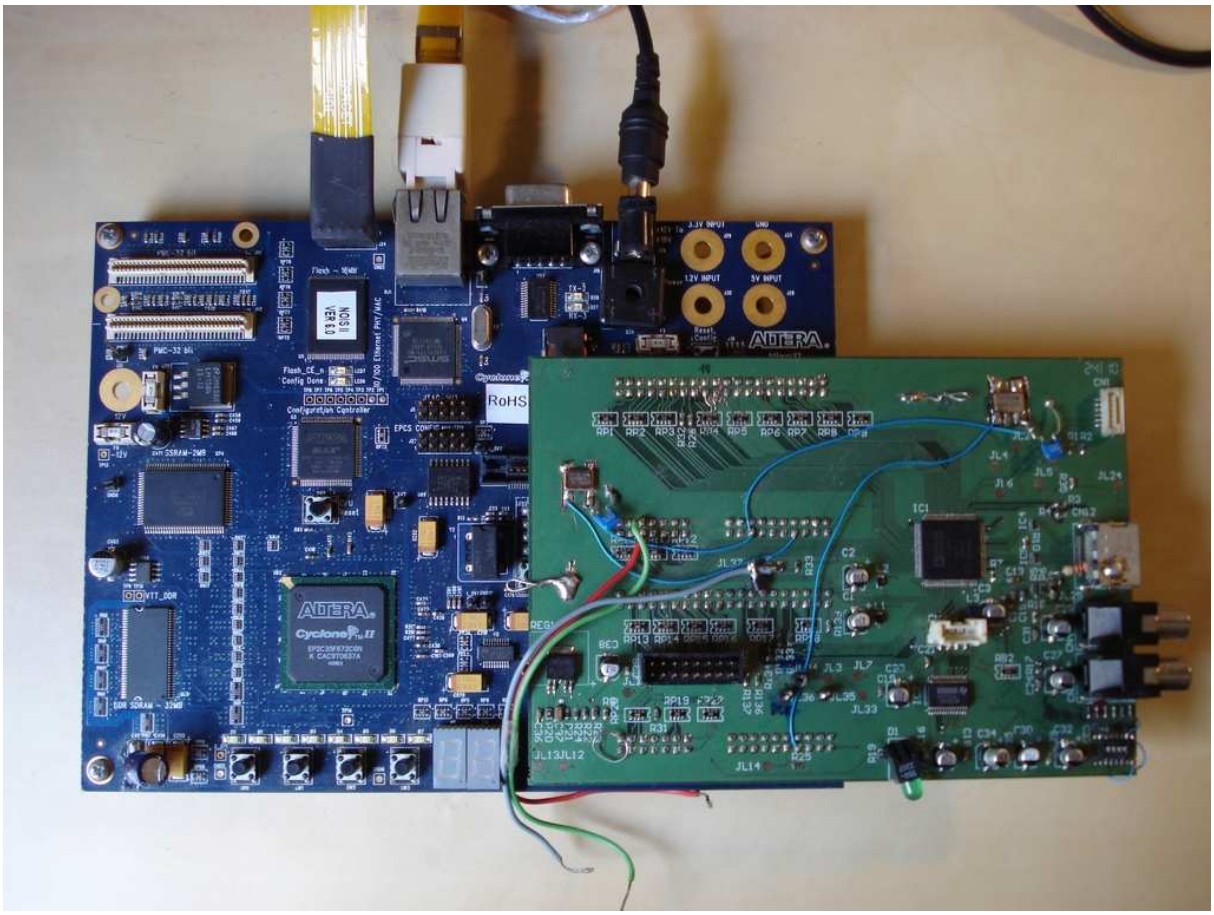


Figure 11.2. Final HDMI generator system after the assembly of the two main parts

The system is then powered and the main power rails +5V, +3.3V and +1.8V are checked again, now with power taken directly from the 2C35 kit expansion connector.

With the assembled final prototype, the next step is to check the signals between the FPGA and the ADV7511 HDMI transmitter, especially audio and video interfaces. Although some cores have been already simulated with ModelSim and sometimes debugged using SignalTap tool, these tests cannot substitute the real on field test, and for this purpose the HDMI Transmitter board was designed with some useful jiglands to check these signals.

Video Interface

As it had been explained in the corresponding chapter, the configuration information in the registers of the video generation core after a reset corresponds to the 720p@60Hz video timing. The digital inputs of the MSO are used in this test setup to check the pixel clock frequency and check the correct timing and synchronization of HSYNC, VSYNC and DEN signals tested in the HDMI transmitter board.
In order to check the video interface, the video signals are checked in the jiglands or connectors listed in Table 11.1.

Signal	Test point to check it	Test detail
PIXEL_CLK	Resistor R22 pads	Check pixel clock frequency is 74,25MHz (720p@60Hz – VIC #4)
DEN	Connector CN9 Pin 7	Check DEN active between HSYNC pulses.
VSYNC	Connector CN9 Pin 5	Check line frequency is 45kHz (720p@60Hz – VIC #4)

HSYNC	Connector CN7 Pin 6	Check frame rate is 60Hz (720p@60Hz – VIC #4)

Table 11.1. List of Video Interface jiglands or Test Points

It is remarkable that it was discarded to check the video data inputs one by one, but during the final system validation some image degradation was appreciated, and after going back to check all data bits one by one eventually it was found that two of the bits were shorted because of a soldering bridge, causing that the generated pattern was not the expected one.

In Figure 11.3 it can be appreciated how the pixel clock runs at the appropriate frequency of 74,25MHz, corresponding to the pixel clock frequency for the 720p@60Hz video timing (VIC #4).

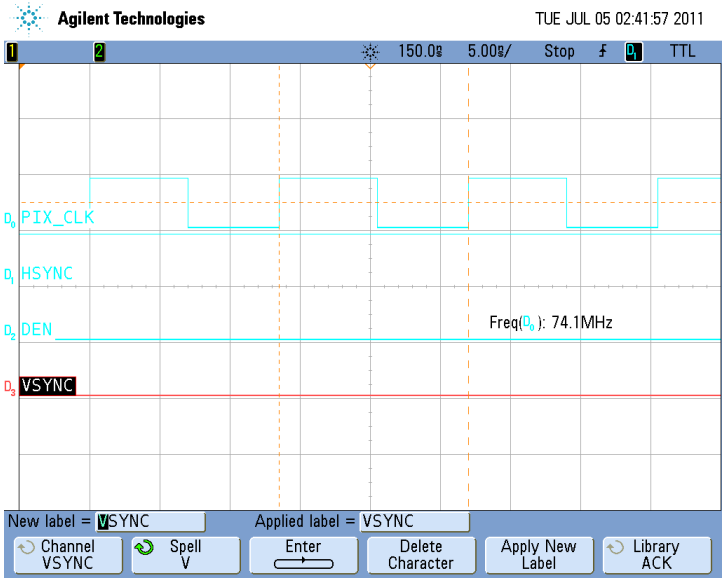


Figure 11.3. Video signals capture, pixel clock running at approximately 74,25MHz

In Figure 11.4 it can be appreciated how the frequency of Hsync signal is 45.00kHz, corresponding to the correct line frequency for the 720p@60Hz video timing. It can also be appreciated how DEN signal is only active after the Hsync pulse and the back porch blanking.

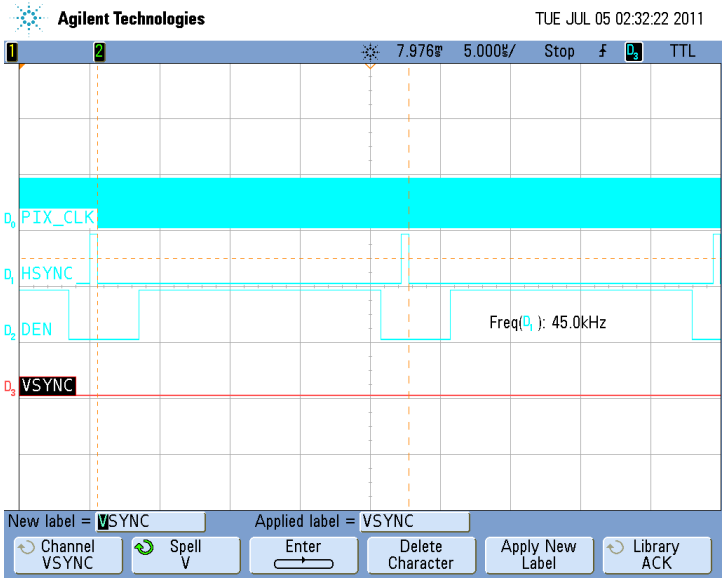


Figure 11.4. Video signals capture, Hsync frequency at 45kHz

In Figure 11.5 it can be appreciated how the frequency of Vsync signal is 60Hz, corresponding to the correct frame rate for the 720p@60Hz video timing (VIC #4).

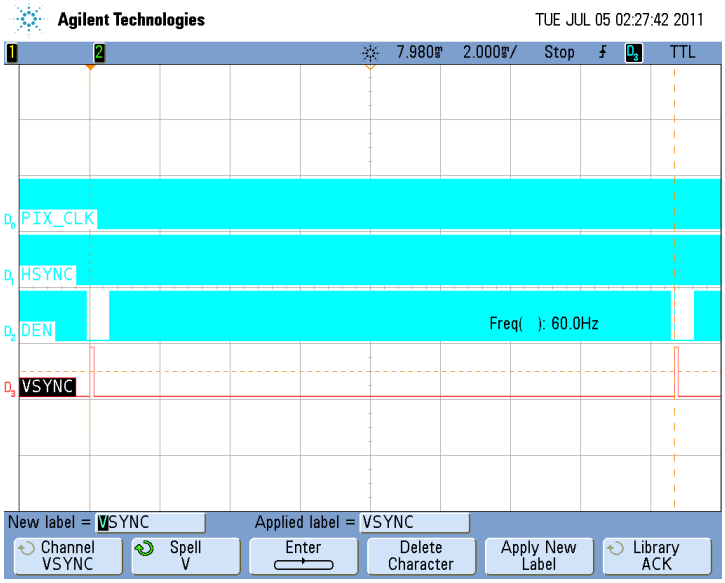


Figure 11.5. Video signals capture, Vsync frequency at 60 Hz, a standard frame rate.

Audio Interface

In the audio case, the configuration information in the registers of the I^2S generation core after a reset is planned to generate an I^2S standard stream with 32 bits of data per channel and the following clock frequencies:

$MCK = 24.576MHz$
$BCK = 12.288MHz$
$LRCK = 192kHz$

In order to check the audio interface the I^2S signals are checked in the jiglands or connectors listed in Table 11.2

Signal	Test point to check it	Test detail
LRCK	Jigland JL35	Check sampling rate frequency is 192kHz
SCK	Jigland JL36	Check bit clock frequency is 12,288MHz
DATA	Connector CN7 Pin 28	Check correct I^2S data format
MCK	Jigland JL34	Check master clock frequency is 24,576 MHz

Table 11.2. List of Audio Interface jiglands or Test Points

In the following captures some features of the I^2S signals are checked in order to assure that design fulfills the expectations and that the ADV7511 is going to correctly recognize the input stream. In Figure 11.6 it can be appreciated how the LRCK clock runs at the sample frequency, in this case 192kHz.

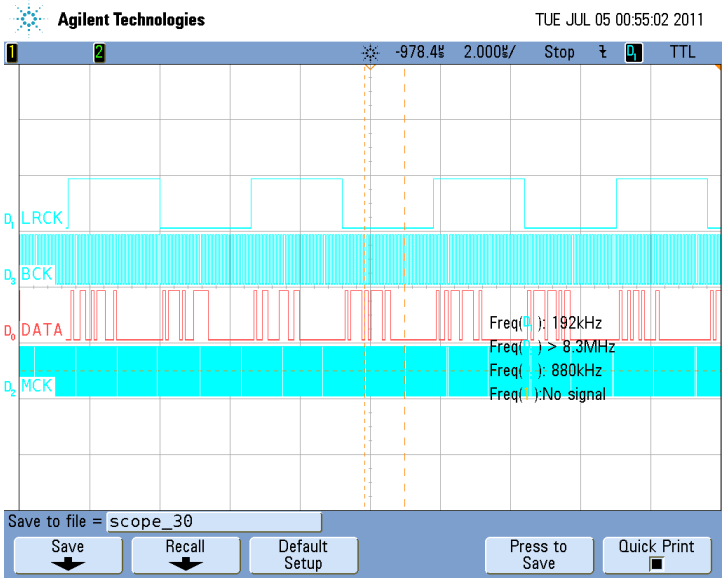


Figure 11.6. I²S signals capture, it can be appreciated how the LRCK runs at 192kHz

In the Figure 11.7 it can be appreciated how BCK and MCK run at the expected frequency, that in this case is exactly 64 times the sampling frequency for BCK, approximately 12,288MHz, and the master clock MCK runs at 128 times the sampling frequency, approximately 24,576MHz. It can also be appreciated that all the clock signals are synchronous, another important requisite to assure whether the ADV7511 recognizes automatically the I²S audio input stream..

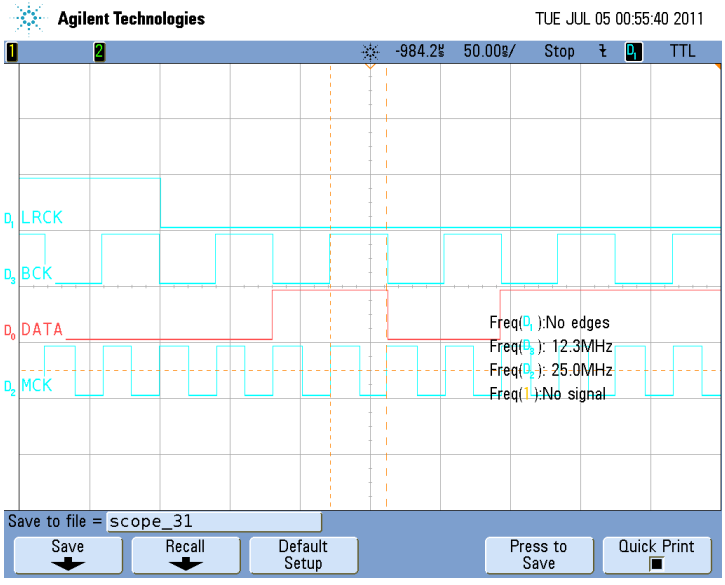


Figure 11.7. I²S signals capture, it can be appreciated how the BCK runs approximately at 12,288MHz and MCK runs at double of this frequency, approximately 24,576MHz.

In the Figure 11.8 it can be appreciated that the signals follow the standard I²S Data format, where the MSB is transmitted just one BCK cycle after a LRCK clock edge. In the figure MSB is transmitted one BCK cycle before the LRCK falling edge that means the start of a left channel sample. It is also checked that sample length is 16 bits, by observing the length of the specific sample in the capture. The reason is that the word size of the sine wave ROM is 16 bits, although there are 32 BCK clock cycles during the period where LRCK remains low before the next rising edge.

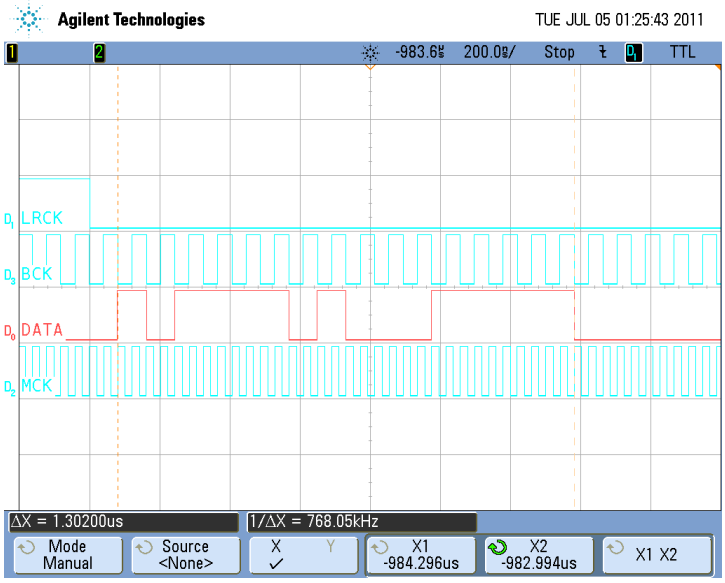


Figure 11.8. I^{2}S signals capture, the correct bit alignment with BCK, and the delay of one Bit Clock Cycle after the LRCK falling edge can be appreciated, as well as the 16-bit length of the audio data sample.

11.2.3 HDMI generator system verification. Use of ethernet control interface to make a deeper check of the hardware

The audio and video interfaces verification has assured that the two main cores are working correctly with the default values after a reset, but to check the configurability and the other parts of the system, the Ethernet control interface must be used to communicate with the board from the host PC.

The board can be directly connected with the host computer NIC using an Ethernet crossover cable. The next step is to configure the host computer NIC to establish a communication with the HDMI generator board. As explained in the *Chapter 10*, the board takes a fixed IP, 10.0.0.51, after the expiration of the DHCP timeout, when no DHCP server is found to automatically get the IP configuration. When HDMI generator board is directly connected to the computer NIC, it always takes the 10.0.0.51 IP address because there is no DHCP server running on the host computer, so the host NIC must be configured with an IP address inside the HDMI generator board IP range in order to have logic access to it.

Once the NIC is configured it is possible to access to the HDMI generator board. As it has been explained HDMI generator board acts as a socket server, and there is a listening socket waiting for incoming connections on port 30. The simplest way for the client host computer to connect to the board is to use a simple Telnet client that provides a bi-directional command-line interface access. From a Windows OS command line, the default Telnet client can be executed with the following command:

```
>> telnet 10.0.0.51 30
```

When HDMI generator board accepts the incoming connection it immediately returns the menu, and remains waiting on the incoming data to process it as a control command, as shown in Figure 11.9. All the detailed information about supported commands for the HDMI generator board can be found in the *Annex G: HDMI Generator control commands list*, so the

syntax of the command is not detailed every time a command is used for the system verification, as it can be easily found in the command reference.

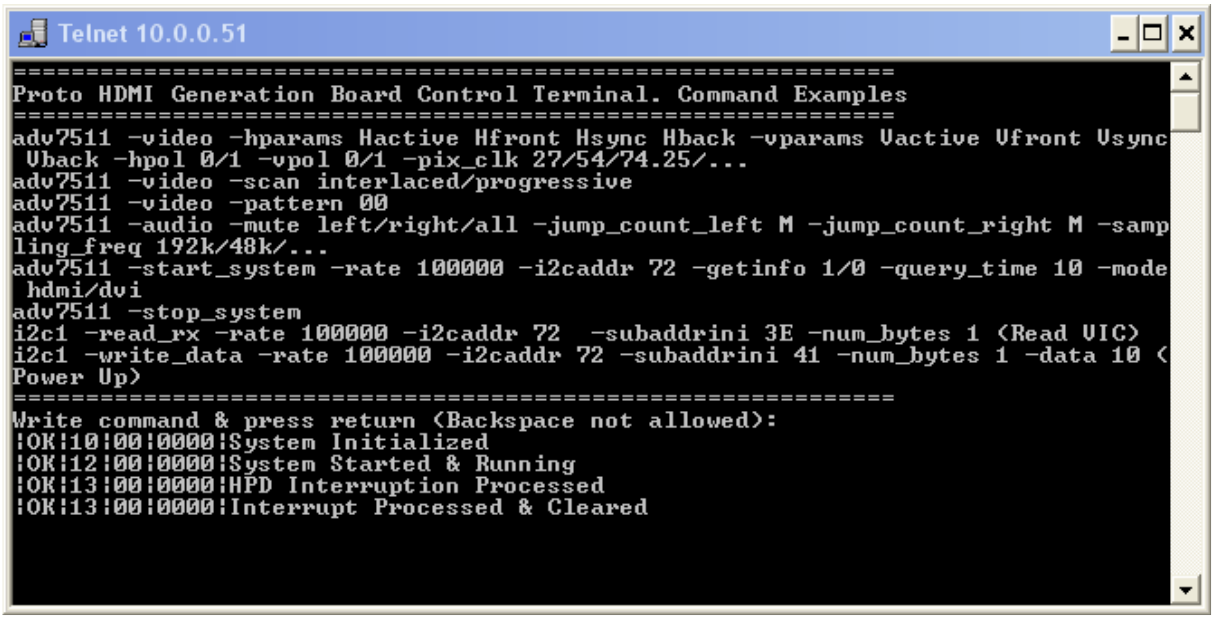


Figure 11.9. Initial menu & system boot-up information in the Windows Telnet client.

I²C bus for communication between µC ↔ ADV7511

Without the help of the ethernet control interface it was impossible to test that part of hardware. The command i2c1 allows direct access to the I²C bus and is very useful to read/write registers of the ADV7511 during debugging phases, as it generates the read/write transfers from master controller to the ADV7511 slave interface and return the data read for the socket in case the transfer finish successfully.

To test the interface for the first time the 16-bit Chip ID register in subaddress 0xF5 is read to check the bus interface is working correctly. As it returns the 16-bit value of the register 0x7511, it can be concluded that the read transfers in the I²C bus work as expected. Figure 11.10 shows the command and its response.

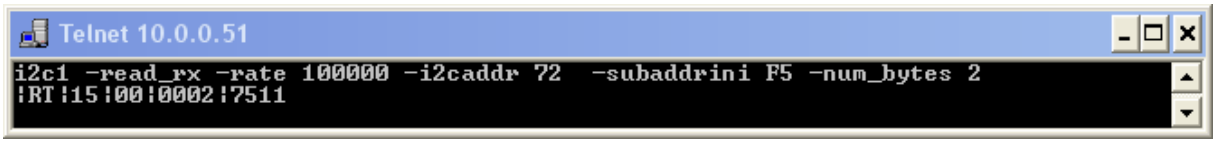


Figure 11.10. I²C read command to read the Chip ID read-only register and its response

In order to test a write transfer, the Power Down register bit (0x41[6]) is written to '0', to power up the ADV7511 and leave the low-power mode. The register write is masked, it means that first the register is read and later is write back with the same contents but forcing bit 6 to '0'. Eventually the register is read again to test that the system has been correctly powered up, but it shows the same contents and this is the correct behavior because the system remains in low-power mode if no HDMI cable is connected, it means the HPD pin remains low. Figure 11.11 shows the three commands sent and its responses.

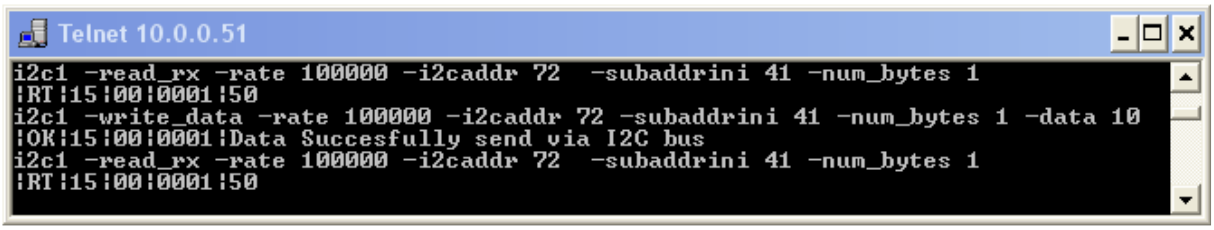


Figure 11.11. Masked I^{2}C write in Power Down register bit (0x41[6]), but system remains in low-power mode

HPD Detection

The HPD pin of ADV7511 is used to detect the presence of an HDMI sink device directly connected to the HDMI connector. The status of the HPD pin can be read in register bit 0x42[6]. The sequence of commands is shown in Figure 11.12, and it can be appreciated how the bit 6 goes from low (0x90) to high (0x50) when the HDMI cable from the sink is connected. Eventually when the HPD pin is high because and HDMI sink is connected to the ADV7511, the system can be powered up (0x41[6]=>0) with the same command tested before and in this case it does not come back to low power mode.

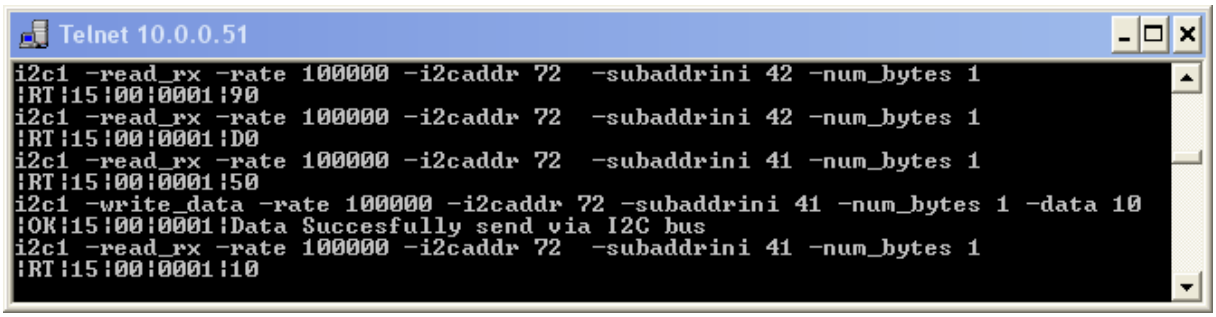


Figure 11.12. HPD pin register bit (0x42[6]) read with cable connected and disconnected, when it goes high ADV7511 is successfully powered up

During power-up it was found a little error on the HPD pin related hardware as the pin was originally left floating, and the state when cable was disconnected was not defined, generating problems because the interruption to start the system was not correctly triggered. A pull-down resistor was added to the HPD line to force the not connected state, in order to reliably differentiate between a floating state, when HDMI sink is not connected and a high-level state when the sink is connected. The rework with an external resistor can be appreciated in the first photo of the HDMI Transmitter board in Figure 11.1, near to the shield of HDMI connector.

RxSense Detection

RxSense refers to the detection of TMDS clock line pull-ups in the HDMI sink. The status of the RxSense detection can be read in register bit 0x42[5]. The sequence of commands is shown in Figure 11.13, and it can be appreciated the state with cable disconnected (0x90), HPD pin=0 (bit6) and RxSense=0 (bit 5), the state when the HDMI cable from the sink is connected (0xD0), HPD pin=1 (bit6) and RxSense=0 (bit 5), and eventually the state when TV is switched to the HDMI input (0xF0), HPD pin=1 (bit6) and RxSense=1 (bit 5).

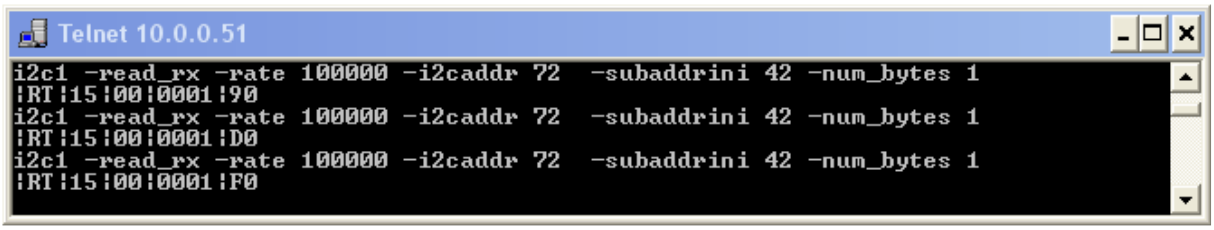


Figure 11.13. Evolution of HPD pin and RxSense detection register bits in different states of interconnection between the HDMI source and sink

Interruption (INT) line

The ADV7511 has interrupts to help with the system design. The interrupts allow the internal HDCP/EDID controller to alert the system of some maskable events. The purpose of this verification is to check that INT line generates a trigger once the enabled event occurs.
The main events that generate an interruption are the HPD pin and RxSense detection. These events are enabled by changing the HPD Interrupt Enable bit (0x94[7]) and the RxSense Interrupt Enable bit (0x94[6]) in the Interrupt Mask register at subaddress 0x94. There are a lot of events that can generate an interrupt, so the interrupt source can be found in the associated Interrupt Status register at subaddress 0x96. In the sequence of commands shown in Figure 11.14, the interrupts for these two events are enabled, and after connecting the HDMI cable to force the interrupt generation, the Interrupt Status register is read to check the source. It results to be the HPD detection (0x96[7]). The interrupt is cleared when the Interrupt Status register is written back with a '1' in the corresponding bit that indicates the source of the interruption, in the example below with a '1' in the HPD interrupt detected bit 0x96[7]).

Figure 11.14. HPD and RxSense detection interrupts enabled by changing the Mask register. After reading the Interrupt Register to check interrupt source, the interrupt is cleared

After checking how to enable the interrupts and read back the interrupt source from the microcontroller, the next step is to validate that interrupt signal is physically triggered when the interrupt occurs. In order to check it, the signal is tested in JL7, and with the previously configured value in the interrupts mask register, a rising edge appears when the HDMI cable from the sink is connected.

Video and Audio Input features detection from ADV7511

The video mode detection feature can inform the user of the CEA-861E Video Identification Code (VIC) of the video sent from the video generation core to the ADV7511. It only detects VICs defined in CEA-861E. This feature only needs user input of the aspect ratio for VICs where this is the only difference, to be able to distinguish between them.
The VIC detected can be read in VIC read-only register (0x3E[7:2]), and the number corresponds to the detected VIC. In case of video being input from FPGA, the configuration

information in the registers of the video generation core after a reset corresponds to the 720p video timing, identified with VIC #4.
In Figure 11.15 it can be clearly appreciated the value read from VIC detection register. To get the detected VIC, the read value of all the byte is right-shifted 2 bits, 0x10>>2=0x04, so the ADV7511 is correctly detecting the video generated from the FPGA.

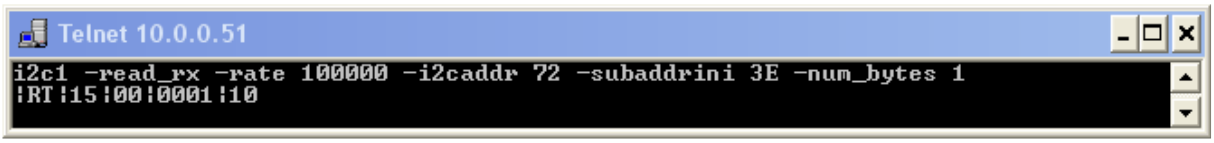


Figure 11.15. VIC detected register (0x3E[7:2]), read with video being input corresponding to 720p-60Hz, VIC #4. The read value have to be right-shifted two bits, 0x10>>2=0x04

For the audio input, the ADV7511 can accept I^2S audio with a maximum sample word length of 32 bits per channel, but it also accepts 16 bits per channel. The ADV7511 will adapt to 16 or 32 bit mode automatically and the detected mode can be read in register 0x42[3].
In case of audio being input from FPGA, the configuration information in the registers of the audio generation core after a reset corresponds to 32 bits per channel although the ROM only has 16 bits per sample, so only the first 16 bits are real information, but the ADV7511 has to detect that there are really 64 bit clock cycles for every LRCK cycle, 32 cycles for every channel. In Figure 11.16 it can be clearly appreciated the value read from I^2S bit mode detection register. The value of third bit is '0' corresponding to 32-bit mode detected.

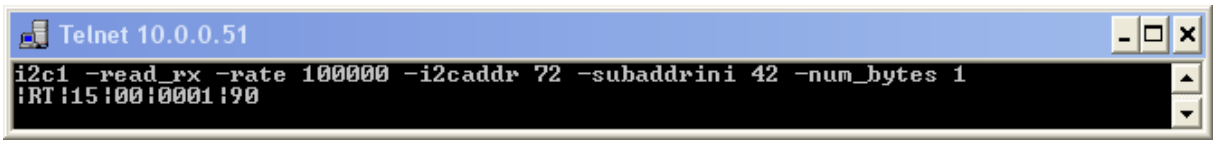


Figure 11.16. I^2S bit mode detection register (0x42[3]), read with audio being input corresponding to 32-bit mode. Third bit value is '0' corresponding to 32-bit mode detected

11.3 Complete stand-alone system validation. Final hardware and firmware verification

All the verification previously made has been oriented to test that all needed input signals, and in general all hardware around the ADV7511 is working as expected. To be able to make all these tests eventually a test firmware with the basic functionalities has been needed, especially with drivers implemented for the I^2C master core, in order to interact with the ADV7511 through its I^2C bus interface from the Ethernet control interface.

As all this previous inspection has been completed the next step is really to test the system with the final user firmware, and to check the final functionalities of the HDMI generator are really working as expected, that means that software enables correctly the HDMI generation system with the default values after a boot-up, and that every configuration change made from the control terminal are really followed.

The test setup in order to validate the correct operation of the system in a normal boot-up consists of all the equipment specified in the Se*ction 11.1 - General test equipment and requirements for verification*. Figure 11.17 shows the block diagram of the test setup.

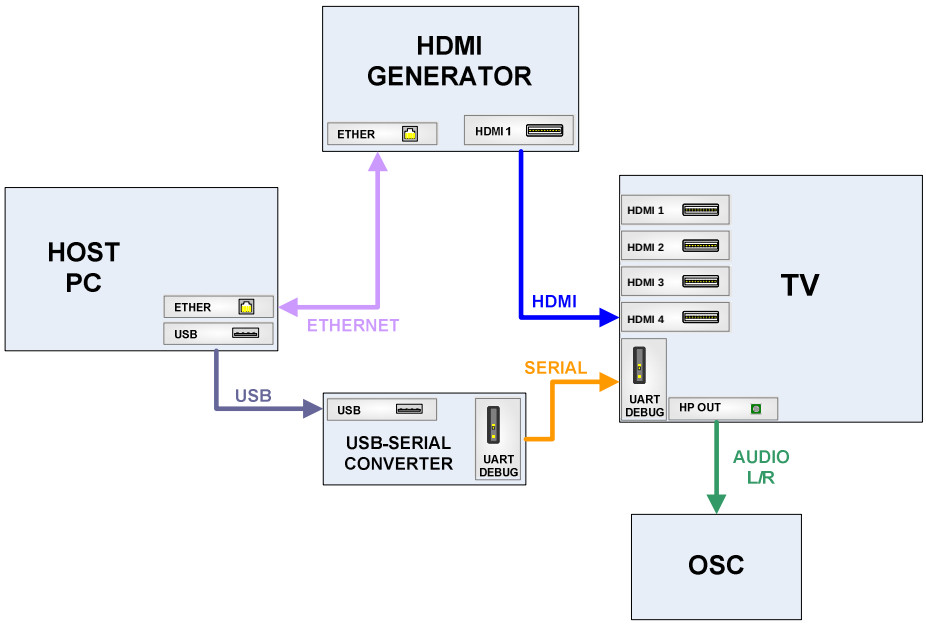


Figure 11.17. Block diagram of test setup for complete stand-alone system validation

The TV in factory mode is used to check all the features of HDMI generator. Basically in factory mode, the TV enables a serial protocol used to query most of the TV internal registers from a Host PC equipped with a serial port. In that case with a functional TV, some of the registers related to the HDMI receiver IC in the TV are queried to check if the HDMI generator works appropriately. Typically the same registers are later used to create test sequences to automate the TV features inspection in production lines. The serial debug interface between Host PC and the TV can be appreciated in the block diagram. The debug interface is useful to check advanced HDMI features like color depth or resolution that cannot be visually appreciated on the screen with a Color Bar pattern. The simpler video and audio tests are made with a picture check in the TV screen, and with an oscilloscope to test the single audio tone in left and right the audio outputs.

11.3.1 HDMI generator stand-alone system boot-up

In a normal boot-up the HDMI generator board has to enable the interruptions of the ADV7511 HDMI transmitter, and the software in the NiosII has to handle all the events in a proper fashion, that means that ADV7511 has to be powered up when a connection with an HDMI sink is detected, and when the EDID information of the sink has been correctly read, the ADV7511 has to make the appropriate audio and video setup to generate a correct HDMI signal according to the current system configuration.

All these events can be monitored with the ethernet control interface that constantly throws information about interrupt processing to the socket. The capture in Figure 11.18 shows a normal boot-up with the HDMI generator disconnected.

1. HDMI generator boots-up normally with the HDMI cable disconnected.
2. The system detects the HDMI cable connection but TV is in standby mode, it means that RxSense detection remains low and ADV7511 remains in low-power mode.
3. When TV is powered, the HDMI generator detects the RxSense interruption but it can be appreciated how the TV brings the HPD low a couple of times in order to force the HDMI generator to automatically read again its EDID. This process to force an EDID

re-read from an HDMI sink, and the general process of read the EDID from the HDMI source in a rising edge of HPD signal is detailed in the HDMI specification. *[20]*

4. After the EDID is successfully read, the EDID ready interruption is processed, the HDMI generator configures audio and video with the default setup, and it activates HDMI TMDS signals.

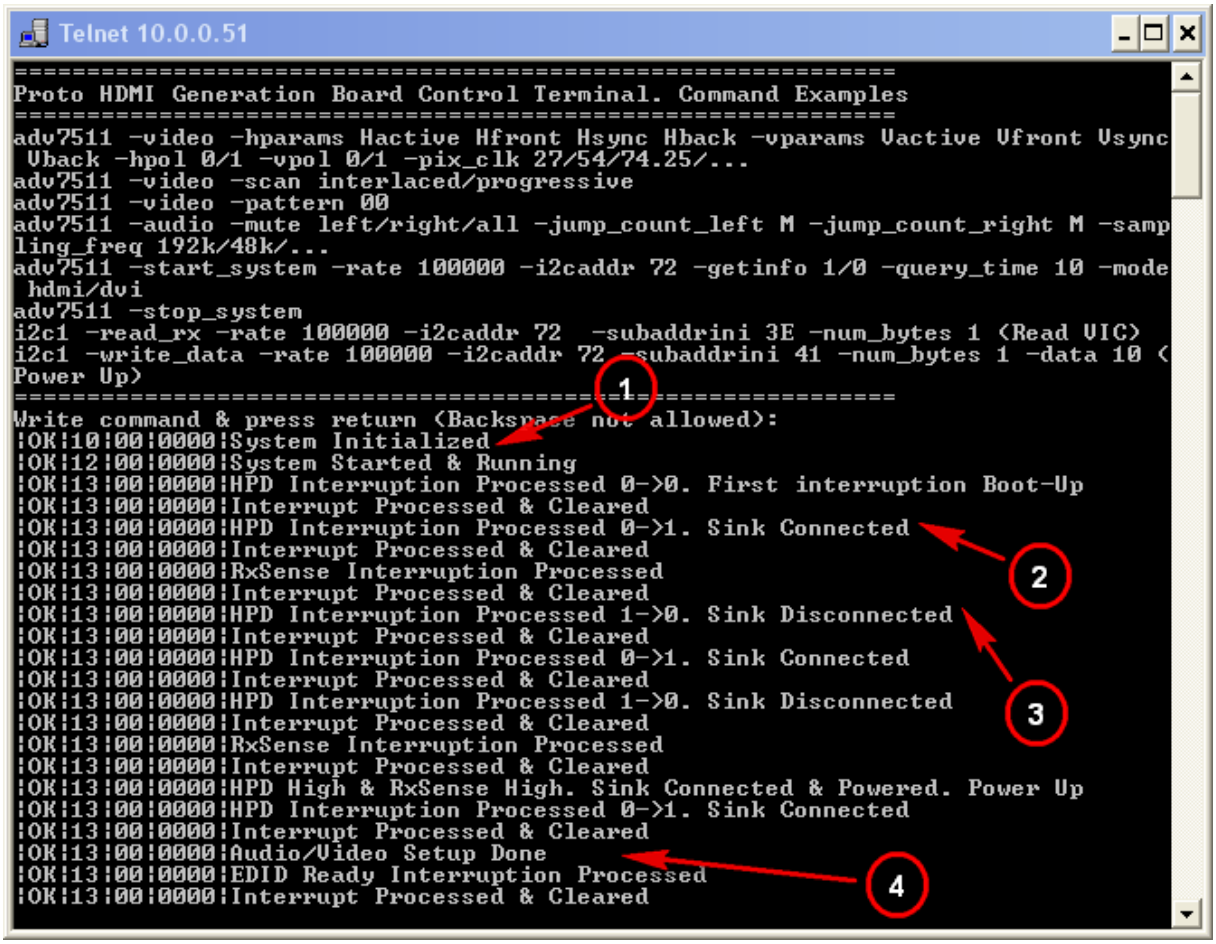


Figure 11.18. System boot-up and successive interrupt processing information in the client terminal

To check the EDID contents read from the TV, the ADV7511 has an independent EDID memory map that contains a mirrored copy of the information read from the TV EDID. To check that the contents has been correctly read, the first 18 bytes of the EDID, corresponding to the fixed Header and the Vendor/Product ID blocks. More detailed and deep information about EDID structure can be found in the provided references *[52]*. The command to read all this information is shown in Figure 11.19 and it can be clearly appreciated the different I^2C slave address, 0x7E corresponding to the special EDID Memory Map instead of the Main Registers Map addressed with 0x72.

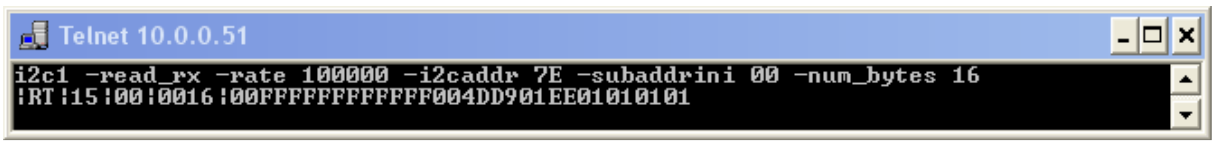


Figure 11.19. First 18 bytes of EDID read with the ADV7511 I^2C master interface to TV

The contents read from the EDID are analyzed with a special EDID Interpretation application provided by Sony to help EDID implementers. The data interpretation for the read bytes is shown in Figure 11.20, and all the contents, like the fixed header pattern, the *SNY* manufacturer name and the 0xEE01 product code, owned by Sony are exactly as expected.

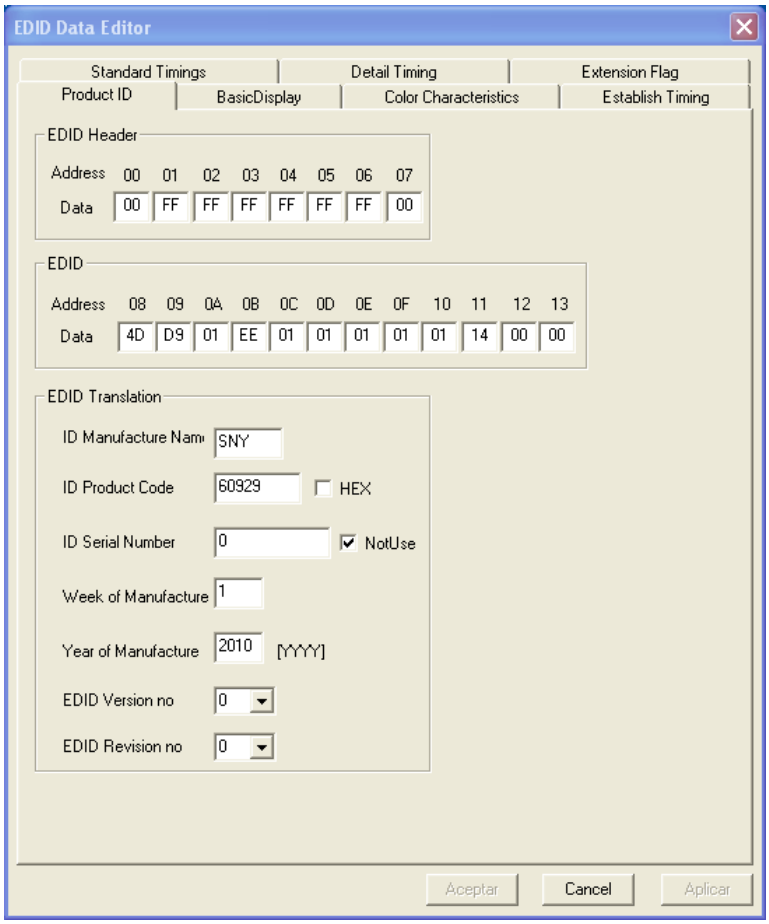


Figure 11.20. Interpretation of data read from the Header and Vendor/Product ID blocks of the EDID corresponding to a Sony TV manufactured at 2010

The capture in Figure 11.21 shows the TV switched to HDMI4 input with HDMI generator board properly connected. The screen picture shows a 100% color bar pattern that is the default pattern generated from the HDMI generator board, the OSD in the TV also shows the detected timing information, 720p, corresponding to the default video setup, so the HDMI picture check result is satisfactory.

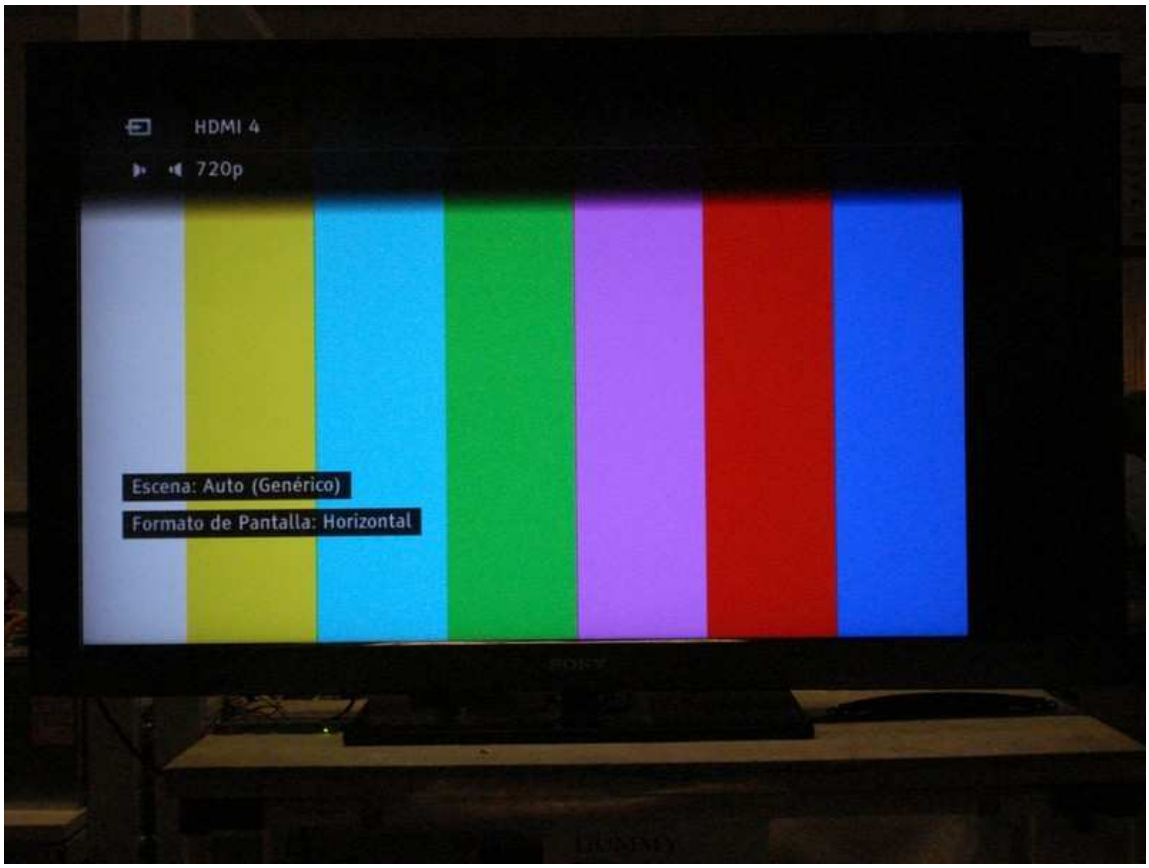


Figure 11.21. Default 720p Color Bar pattern from HDMI generator in test TV screen

Although the two audio tones for left and right channels can be appreciated listening the TV speakers, in order to check it in detail, the TV headphones audio output is connected to the two oscilloscope channels with a headphones Jack → 2 BNC cable.
The Figure 11.22 shows the two sine wave waveforms of left and right channels when TV switched to HDMI. The frequencies of the two sine waves in every channel are exactly as expected, corresponding to the default audio setup, a 400Hz audio tone for left channel and a 1kHz audio tone for the right channel

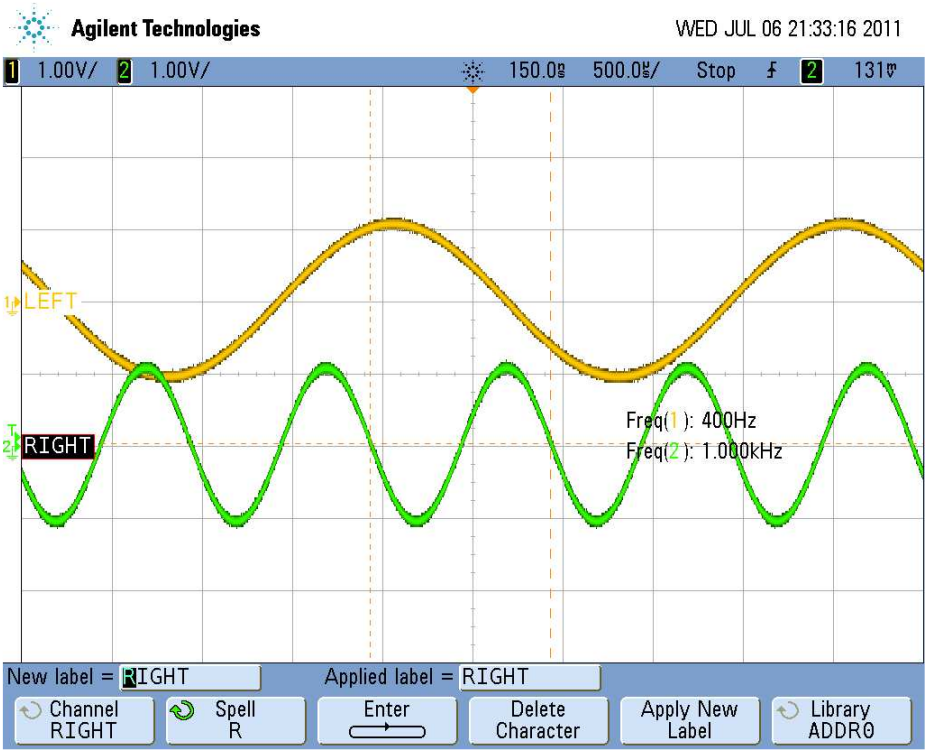


Figure 11.22. Headphones Left and Right channel when TV is switched to HDMI with signal from HDMI Generator.

When the HDMI cable is disconnected, the system also has to detect the interruption and the ADV7511 has to go back to the low-power mode, the interruption processing information can be appreciated in the Figure 11.23.

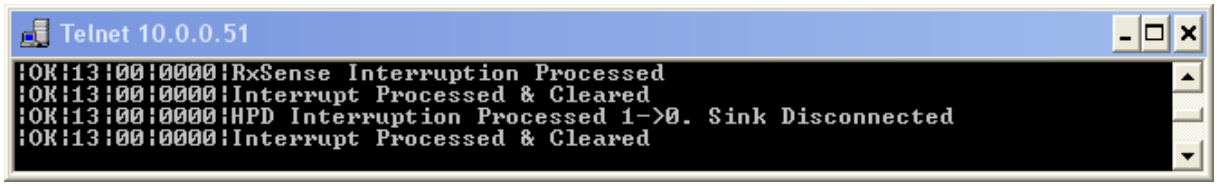


Figure 11.23. System automatically processes the interruption because of HDMI cable disconnection, and ADV7511 is put back to low-power mode

The Figure 11.24 shows a sequence of two commands, the first one to stop the system, and the second one to activate the cyclical ADV7511 information update in the client terminal interface. It can be appreciated how the information about the board status and the HDMI link is shown on the terminal.

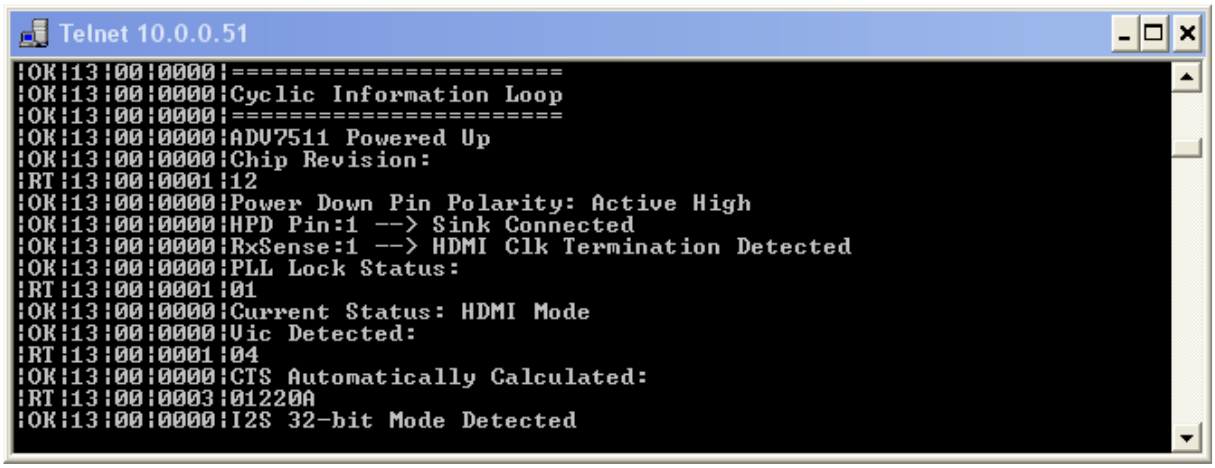


Figure 11.24. Cyclical debug information of the system in the control interface for debug

After all the tests it can be assured that HDMI Generator board software follows the correct flow of events and the stand-alone system is capable of smartly power-up depending on the connection state.

11.3.2 HDMI generator features validation with a TV in factory mode

Eventually, the main application of the HDMI generator is the capacity to configure a big amount of different video formats, by changing the video timing information and the aspect ratio of the incoming video signal to the ADV7511. There is also the possibility to change the color depth from normal 8-bit modes to special deep color modes with 10 or 12 bits.

To finally validate the prototype correct operation, the best test platform is the TV in factory mode, because the debug interface can be very useful to check advanced HDMI features like color depth or resolution that cannot be easily appreciated with a simple Color Bar pattern. The debug interface allows complete access to the HDMI receiver block of the TV, and there are a lot of internal TV registers that can be checked easily with host PC debug software provided by Sony, shown in Figure 11.25, that is used to read/write the TV registers. The main purpose of this application is to help create inspection test sequences, based on reading/writing this registers to inspect the TV features correct operation, but the use given here is the opposite one, to check the test equipment features with a reliable and functional TV.

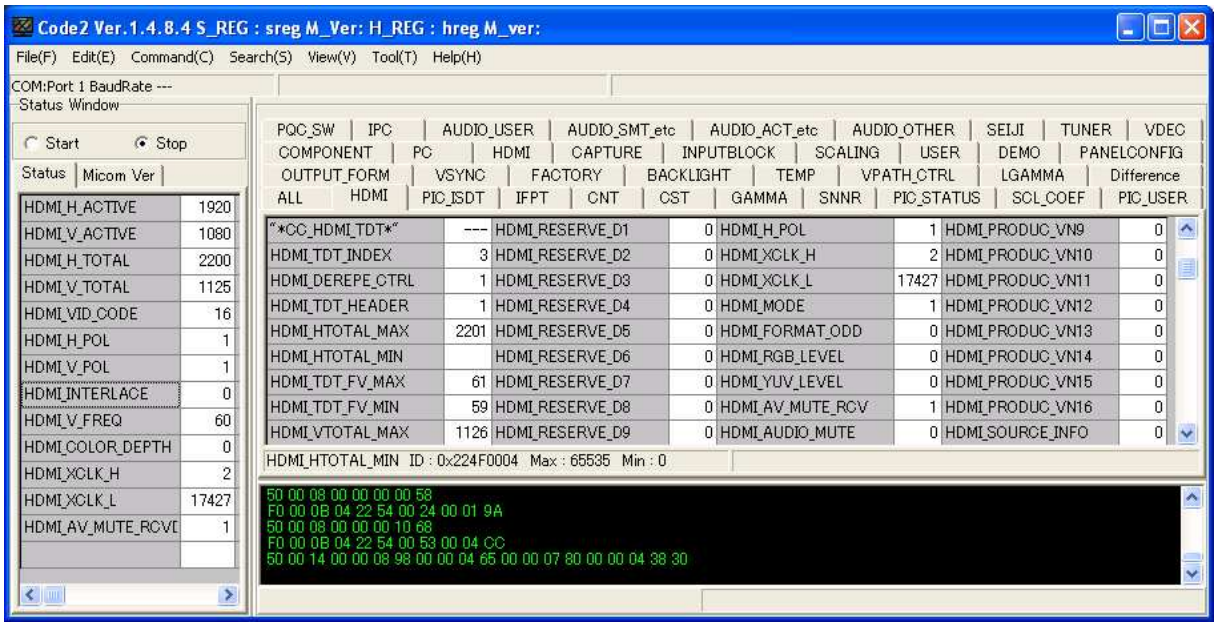


Figure 11.25. Software to read/write TV internal registers used to check HDMI generator special features

The Table 11.3 shows the list of TV software registers (SREGs) used to check the HDMI generator features. The control software allows a set of registers to be monitored at regular times in the left window of the application, so this feature will be used to check the changes that the TV HDMI receiver detects.

Software Register Name	Description
HDMI_VID_CODE	Video Identification Code send in the AVI InfoFrame. The list of VICs can be found in the CEA-861E specification. *[19]* Typical ones tested in the HDMI generator board: • 2: 720x480p@60Hz - 4:3 (Only difference to distinguish VIC is aspect ratio) • 3: 720x480p@60Hz - 16:9 (Only difference to distinguish VIC is aspect ratio) • 4: 1280x720p@60Hz - 16:9 • 5: 1920x1080i@60Hz - 16:9 • 16: 1920x1080p@60Hz - 16:9 • 17: 720x576p@60Hz – 4:3 (Only difference to distinguish VIC is aspect ratio) • 18: 720x576p@60Hz – 16:9 (Only difference to distinguish VIC is aspect ratio)
HDMI_H_TOTAL	Number of total (active+blanking) pixels detected
HDMI_V_TOTAL	Number of total (active+blanking) lines detected
HDMI_H_ACTIVE	Number of active pixels detected, horizontal resolution.
HDMI_V_ACTIVE	Number of active lines detected, vertical resolution.
HDMI_INTERLACE	Interlaced/Progressive flag: 0: Progressive 1: Interlaced
HDMI_V_FREQ	Vertical sync frequency or frame rate detected.

HDMI_COLOR_DEPTH	Color Depth 0: 8-bit 1: 10-bit 2: 12-bit
HDMI_XCLK_H HDMI_XCLK_L	TMDS Clock Frequency Detected (4 bytes) TMDS_CLK = HDMI_XCLK_H * 0x1000 + HDMI_XCLK_L
HDMI_AV_MUTE_RCV	AV Mute request received from the source. 1: AV Mute enabled 0: AV Mute disabled

Table 11.3. List of SREGs used to check HDMI generator features

The main feature of the HDMI generator is the ability to change between a lot of different video formats using the ethernet remote control interface. There is also important to remark the possibility to change the frequency of the left and right audio single tones independently. To test this video reconfigurability some of the typical video formats have been tested. It also has been tested the deep color mode capabilities, as well as the audio tones reconfigurability.

VIC#4: 1280x720p@60Hz – 16:9
Default video format after HDMI generator boot-up. Figure 11.26 shows the picture check and the SREGs of the factory mode TV with that HDMI signal.

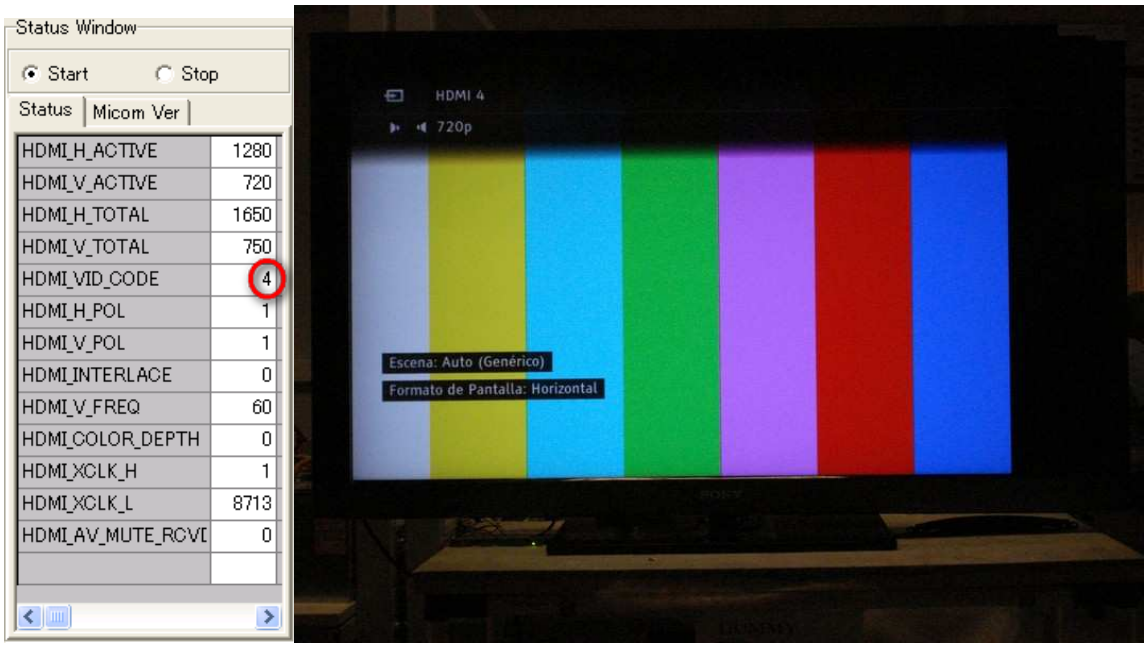


Figure 11.26. TV soft registers values and picture check in 1280x720p@60Hz - 16:9

VIC#3: 720x480p@60Hz – 16:9
Figure 11.27 shows the command to change to this video format. Figure 11.28 shows the picture check and the SREGs of the factory mode TV with that HDMI signal.

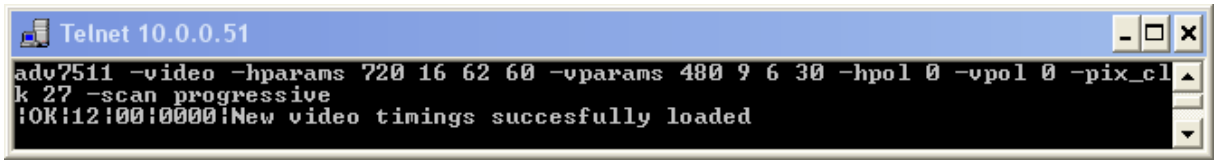


Figure 11.27. Command to change to 720x480p@60Hz – 16:9

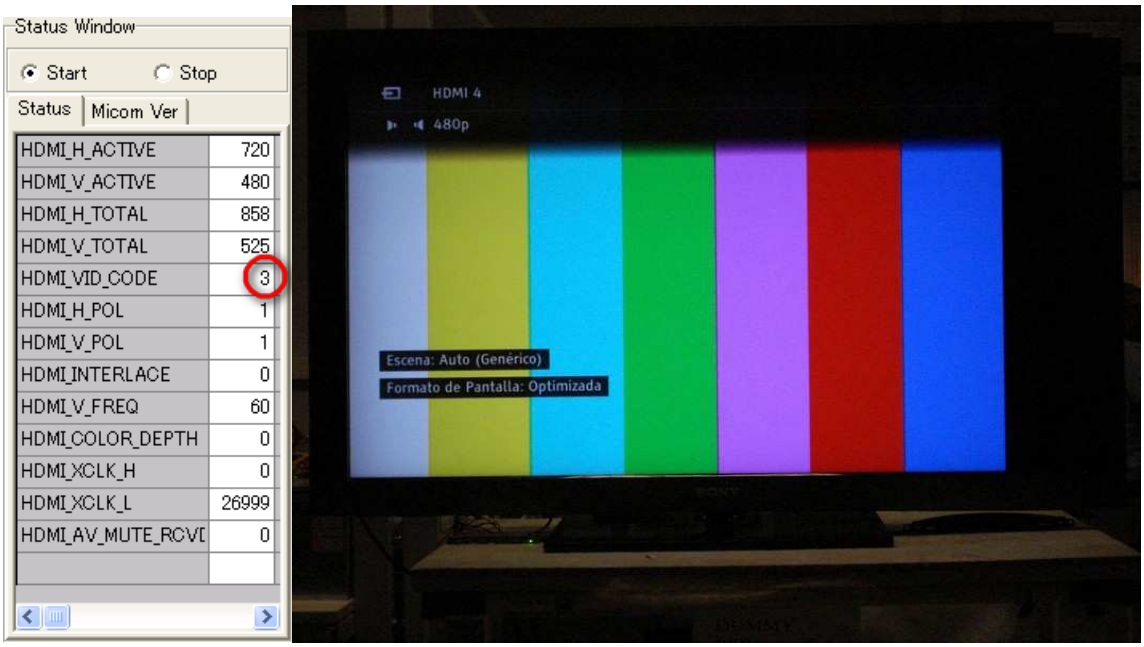


Figure 11.28. TV soft registers values and picture check in 720x480p@60Hz – 16:9

VIC#2: 720x480p@60Hz – 4:3

Figure 11.29 shows the command to change to this video format, with exactly the same timings but different aspect ratio. Figure 11.30 shows the picture check and the SREGs of the factory mode TV with that HDMI signal. It has exactly the same settings than the other timing but different VIC, because the TV receives correctly the aspect ratio information in the AVI InfoFrame packet.

Figure 11.29. Command to change to 720x480p@60Hz – 4:3

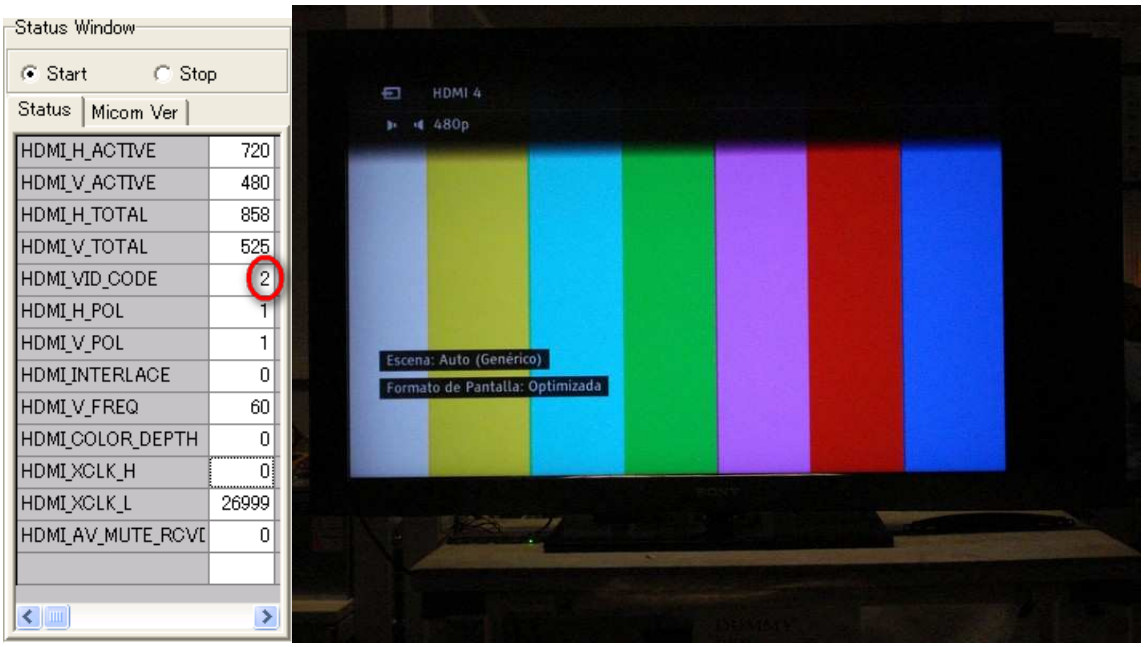


Figure 11.30. TV soft registers values and picture check in 720x480p@60Hz – 4:3

VIC#18: 720x576p@60Hz – 16:9

Figure 11.31 shows the command to change to this video format. Figure 11.32 shows the picture check and the SREGs of the factory mode TV with that HDMI signal

Figure 11.31. Command to change to 720x576p@60Hz – 16:9

Figure 11.32. TV soft registers values and picture check in 720x576p@60Hz – 16:9

VIC#5: 1920x1080i@60Hz – 16:9

Figure 11.33 shows the command to change to this video format, with the particularity of the interlaced scanning. Figure 11.34 shows the picture check and the SREGs of the factory mode TV with that HDMI signal. It is appreciated how the TV detects the interlaced scanning in the special software register, and half of the active lines in every frame.

Figure 11.33. Command to change to 1920x1080i@60Hz – 16:9

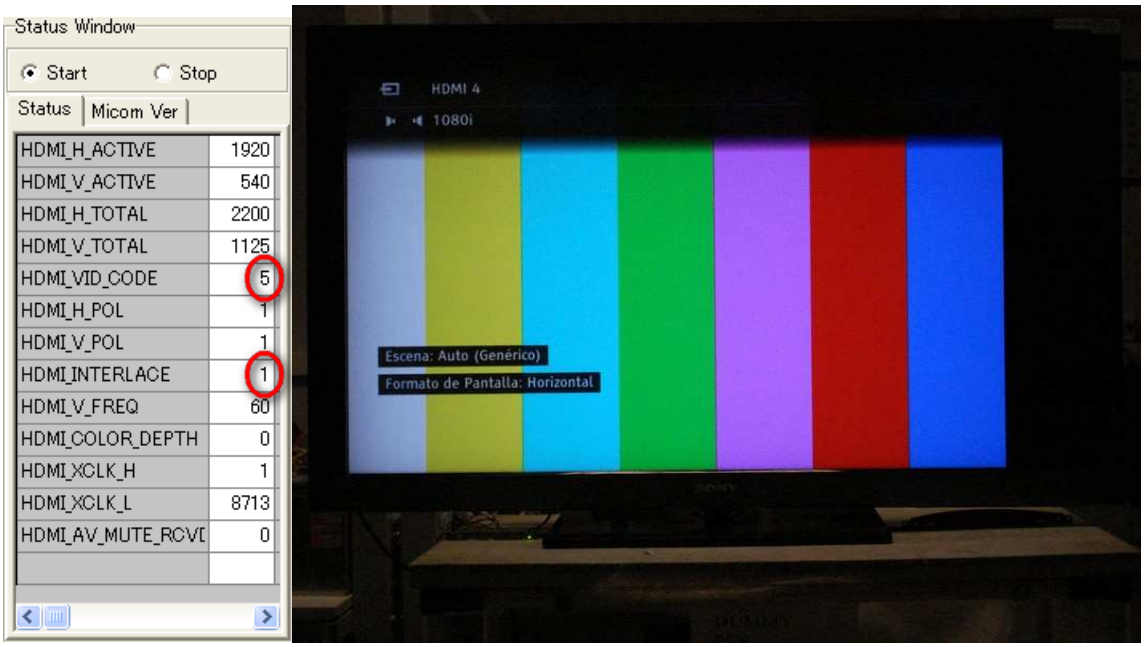


Figure 11.34. TV soft registers values and picture check in 1920x1080i@60Hz – 16:9

VIC#16: 1920x1080p@60Hz – 16:9

Figure 11.35 shows the command to change to this video format, the typical standard of high-resolution video. Figure 11.36 shows the picture check and the SREGs of the factory mode TV with that HDMI signal

Figure 11.35. Command to change to 1920x1080p@60Hz – 16:9

Figure 11.36. TV soft registers values and picture check in 1920x1080p@60Hz – 16:9

Color depth changes to Deep Color modes (1080p with 10-bit, 12-bit)

Figure 11.37 shows the successive command to change the color depth from normal 8-bit mode to 12-bit mode. Figure 11.38 shows the SREGs of the factory mode TV with that HDMI signal. It can be appreciated how the TV detect the color depth change notified in the HDMI signal with the AVI infoframe packet, but also it detects the different TMDS clocks, with multipliers x1.25 and x1.5 from the original 148.5MHz clock of 1080p video timing, needed to packetize all the extra deep color information.

In 8-bit mode, TMDS Clock = HDMI_XCLK_H * 0x1000 + HDMI_XCLK_L = 2 * 0x1000 + 17427 = 148499 → F_{TMDS}=148.5MHz

In 10-bit mode, TMDS clock = 2 * 0x1000 + 54552 = 185624 → F_{TMDS}=185.625MHz

In 12-bit mode, TMDS clock = 3 * 0x1000 + 26140 = 222748 → F_{TMDS}=222.75MHz

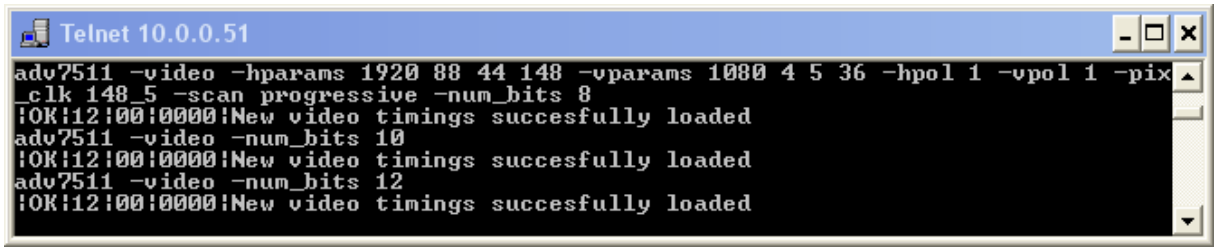


Figure 11.37. Command to change to 1920x1080p@60Hz – 16:9 and successively change the color depth from 8-bit to 12-bit mode

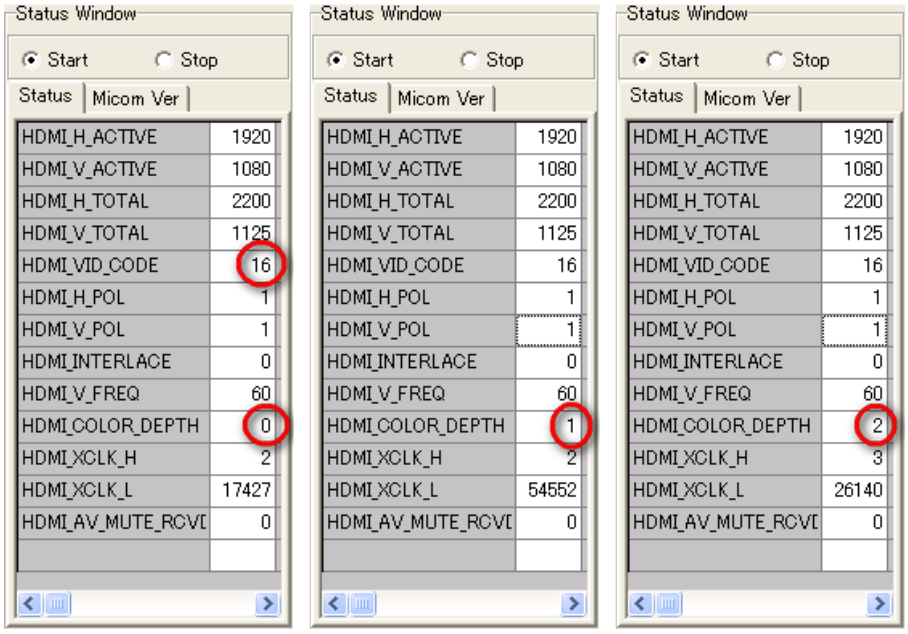


Figure 11.38. TV soft registers values when changing HDMI signal color depth

AV Mute

Figure 11.39 shows the command to activate AV Mute. This feature is used to notify the sink that the source is going to change some of the video or audio signals features. If the sink implements it, the audio and video are muted, and the possible audio or video artifacts are not noticed by the final user.

In the HDMI generator board sometimes can be useful to send an AV mute before changing audio or video properties. Figure 11.40 shows the SREGs of the factory mode TV after sending HDMI packets with AV mute enabled.

Figure 11.39. Command to send AV Mute requests without changing source HDMI signal

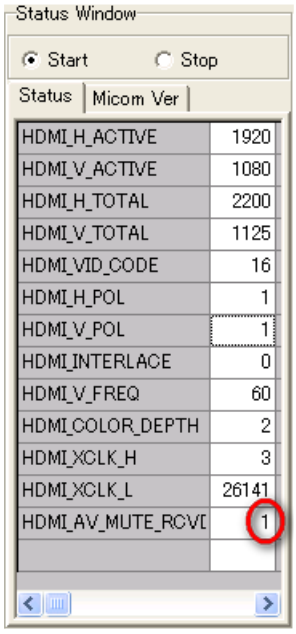


Figure 11.40. TV soft registers values when AV Mute request is received

Audio frequency tones changes for left/right channels
The frequency of the audio single tone in the left channel has been changed between several values in order to check the frequency reconfigurability of the audio single tone in every channel.

The frequency of the audio outputs, depends on the configured sampling frequency that is 48kHz, the more typically supported in any HDMI sink, and the configured ROM jump size The step size values as explained in the chapter 9.4 is selected according the formula:

$$f_{out} = \frac{M \times f_{in}}{L} = \frac{JUMP\_SIZE \times SAMPLING\_FREQUENCY}{1920} = \frac{JUMP\_SIZE \times 48000}{1920}$$

In Table 11.4 there is the list of audible single tones checked.

Jump Size	Output Frequency (f_{out})	Usage
16	400Hz	Typical frequency for audio tests.
40	1kHz	Typical frequency for audio tests.
400	10kHz	Typical high frequency to test audio tweeters, special high-frequency speakers.

Table 11.4. Typical output frequencies for audio tests

Figure 11.41 shows the command to mute the left channel, and the successive values to configure the typical frequencies used in audio tests, 400Hz, 1kHz and 100kHz. Figure 11.42 shows the left channel muted, Figure 11.43 shows the left channel configured at 400Hz, Figure 11.44 shows the left channel configured at 1kHz and Figure 11.45 shows the left channel configured at 10kHz.

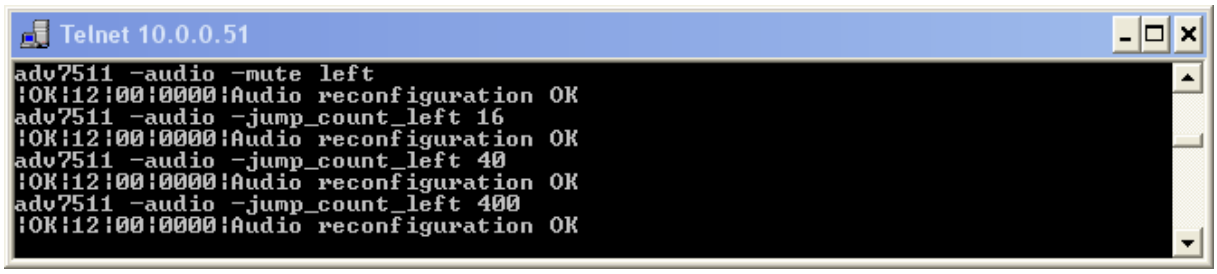


Figure 11.41. Command to mute the left channel and configure it successively to 400Hz, 1kHz and 10kHz

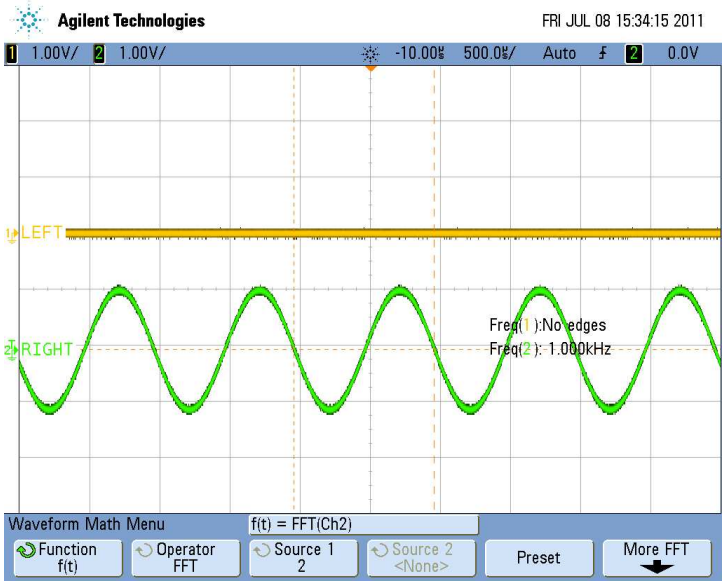


Figure 11.42. Left channel muted

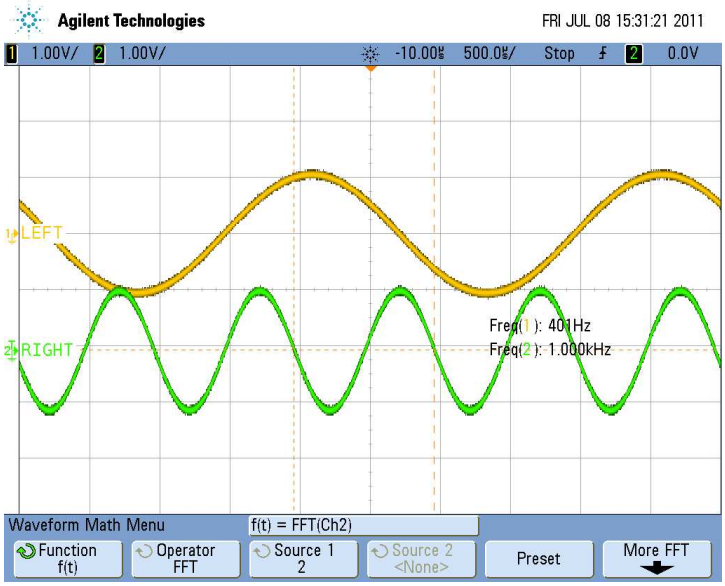


Figure 11.43. Left channel frequency at 400Hz

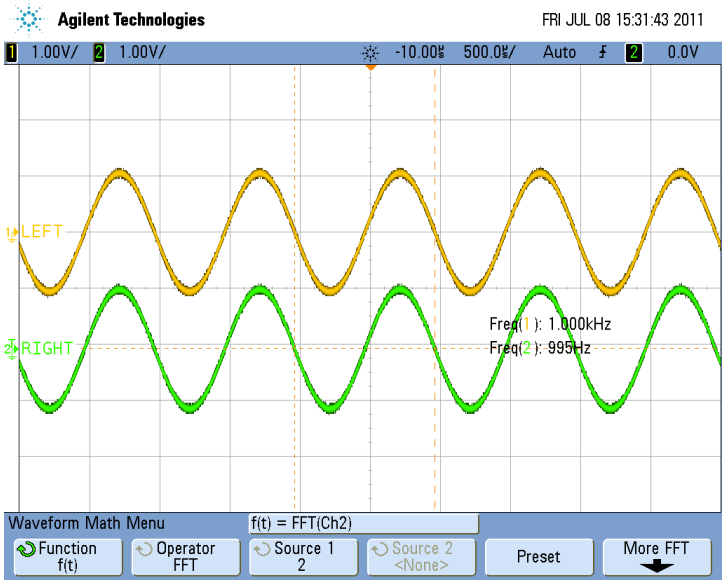


Figure 11.44. Left channel frequency at 1kHz

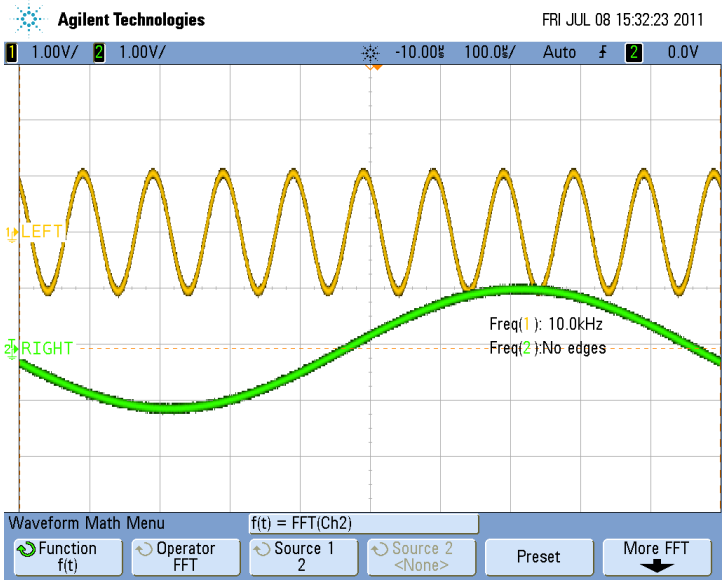


Figure 11.45. Left channel frequency at 10kHz

Chapter 12

Host PC control software development

In the previous chapter it has been introduced how to control the HDMI generator board using the ethernet remote control interface with a simple Windows Telnet client, and although this use is appropriate for power-up, debugging and system validation, this is not the best way to communicate with the HDMI generator board in a production environment.

This chapter introduces the development of a control library to communicate with the HDMI generator board in the production environment. It also introduces the development of demonstration and debugging control software that uses the previously introduced library to be able to interact graphically with the control board, in a more user-friendly way than the remote control text-mode terminal.

12.1 Control library for test automation

12.1.1 Necessity of control libraries

In the production environment, and specifically in the functional test systems, the text-mode Telnet terminal used intensively in the last chapter is not appropriate to change the HDMI generator board modes of operation.

Usually all the functional test systems use ready-to-run test management software designed to help the engineers to develop automated test and validation systems in a faster and simpler way. The test sequences developed with these test management suites usually offer the possibility to integrate code modules written in nearly any programming language and also provide. The common test management software used in BCN Tec is National Instruments TestStand, which supports code modules written in a lot of programming languages.

In this context, it is easy to understand the necessity to develop a control library that will be integrated with the test management software, because that software also controls all the other equipment, hardware and the DUT itself in the functional test system.

So as a rule, every new piece of test hardware with a remote control interface, must provide a code module to be able to use it in the test management software in a different abstraction layer, totally independent of the low-level technology used to implement the remote terminal.

12.1.2 EthernetControl class library development

The HDMI generator board remote control interface is basically defined as a stream socket server bound to a particular IP address and port, that is perpetually waiting for incoming data. In server side when data arrives the appropriate routine is called to either accept/reject a connection request, or process incoming data.

There are some points of the socket server implementation that are important to remark in order to develop the control library that will act as a client:

- This sockets server implementation only accepts one single connection at a time, so any incoming request is rejected while the first connection is open.

- The server use connection-oriented sockets, also called stream sockets, that provide a reliable and guaranteed way to communicate, as the TCP protocol is used to ensure that data is confirmed to be delivered to the destination point. That also assures data packets are received in the same order they were sent out on both directions.
- When a connection is established the server is constantly receiving incoming data until a line feed (LF) character is received. In ASCII is defined as 0x0A, 10 in decimal. Anyway, all the command must be terminated with the combination of ASCII carriage return + line feed, CR+LF (0x0D 0x0A), and it is important to remark that the server is not tolerant to a lone LF, because last byte of data would be discarded, leading to a misinterpretation of the command, defined as all the data previous to the CR+LF termination.
- The command responses from the server follow a pattern in order to be able to automate the correct interpretation in the client side. It will allow to know if the command has been correctly executed or not without errors. The command response contains the following information enclosed by separator strings.
  `|Result|TaskCode|SubTaskCode|ErrorCode/NumBytes|Message/Data`
  - Result: **OK** in case of successful execution of command, **NG** in case of an error, and **RT** in case of a command response containing data.
  - TaskCode: the priority number of the taske executed by the command.
  - TaskSubCode: to provide extra information about the task, but typically unused.
  - ErrorCode/NumBytes: provides a fixed error code in case of NG response, or the number of data bytes in case of a RT response.
  - Message/Data: provides a string description of the error in case of NG response, and the data returned from the board in case of a RT response to a query command.
- When a line termination is received, all the previous data is moved to a different task that parses the command and if the command is correctly recognized and accepted executes the task related to the command. It means that the receiving task is always constantly receiving data and putting it to the execution queue when a line termination is received, so other commands can be processed and executed while a long execution time command is still running. In conclusion, the HDMI Generator board allows concurrent execution of commands. It can be treated in the control library in two ways,
  - Separate the command sending and the command response reception in two methods to allow multitasking.
  - Create methods that wait after a command sending until the command response is received. It effectively discards the use of the multitasking feature but simplifies the development. As all the commands accepted by the HDMI generator board are completed in a very short time, this option has been taken as the multitasking feature is never going to be used in production.

With all these important points in mind, the framework and the programming language have to be selected from all the accepted by TestStand. The .NET framework API option, used with C# object-oriented programming language, has been selected over other options like LabWindows/CVI, LabView or C++ because of all the useful classes for networking that the .NET framework provides. The System.Net.Sockets.Socket class can be used as a socket in a server application as well as in a client application and is extremely flexible as it provides both synchronous and asynchronous operations, because all the I/O calls have a blocking (synchronous) and non-blocking (asynchronous) version. In order to simplify the class library development it has been decided to use synchronous sockets, it means that all the methods

used to send commands to the HDMI generator board block the program execution until data has been completely send or the timeout has expired. The same is applied for the command response that is received from the HDMI generator board. That decision effectively simplifies the development, and can be applied because it has been decided to not use the multitasking feature, that allows several commands to be executed at the same time. To use that feature the socket calls must be asynchronous, it means that two commands could be sent at the same time, and a separate thread would be responsible to get the command responses to every command.
More detailed information about programming with TCP/IP sockets using the .NET framework and C# can be found in the provided references *[53][54][55]*.

Once all the objectives and constraints of the control library have been explained, in this section the main methods of the EthernetControl class library are introduced and briefly discussed. Apart from the methods to connect/close the socket, the other ones basically send a command to the socket and wait until the command response to return.

- bool **ConnectWithEthBoard** (out string errorMsg, string ipString, int ipPort, bool flushMenu, int timeout_ms)

Generic method to establish a connection to a socket server running at the selected IP address and attached to the selected port. Once the socket connection is established, other methods can use the socket to send configuration commands to the remote control socket server. If socket server sends menu information once the connection is established, it can optionally be flushed to not interfere the normal operation.

- bool **CloseConnectionWithEthBoard** (out string errorMsg)

Generic method to close the active socket connection and release related resources.

- bool **SetHdmiVideoSetupEthBoardTS** (out string errorMsg, int Hactive, int Hfront, int Hsync, int Hback, bool HparamsDefined, int Vactive, int Vfront, int Vsync, int Vback, bool VparamsDefined, int Hpol, int Vpol, bool bSyncPolarityDefined, PixelClk pixClk, bool bPixClkDefined, ScanType scanType, bool bScanTypeDefined, int numBitsColor, bool bColorDepthChange, AspectRatio aspectRatio, bool bAspectRatioChange, int patternNumber, bool bPatternChange, int recv_timeout_ms, ref long real_elapsed_time)

Method to configure video setup in HDMI generator board.

- bool **SetHdmiAudioSetupEthBoardTS** (out string errorMsg, SamplingFrequency SamplingFreqMode, bool SamplingFreqDefined, MuteCondition MuteChannels, bool MuteDefined, int JumpCountLeft, bool JumpCountLeftDefined, int JumpCountRight, bool JumpCountRightDefined, int recv_timeout_ms, ref long real_elapsed_time)

Method to configure audio setup in HDMI generator board.

- bool **SetHdmiAudioSetupEthBoardTS** (out string errorMsg, SamplingFrequency SamplingFreqMode, bool SamplingFreqDefined, MuteCondition MuteChannels, bool MuteDefined, double LeftChannelFrequency, bool JumpCountLeftDefined, double RightChannelFrequency, bool JumpCountRightDefined, int recv_timeout_ms, ref long real_elapsed_time)

Method to configure audio setup in HDMI generator board.

- bool **ReadI2cIndividualBytesEthBoard** (out string errorMsg, string i2c_selected, int i2c_rate, byte i2c_address, int subaddr_ini, int bytes_subaddr, int bytes_to_read, out byte[] recv_data)

Method used to read data from an I2C slave device.

- bool **SendI2cIndividualBytesEthBoard** (out string errorMsg, string i2c_selected, int i2c_rate, byte i2c_address, int subaddr_ini, int bytes_subaddr, int bytes_to_send, byte[] tx_data, int recv_timeout_ms, ref long real_elapsed_time)

Method used to send data to an I2C slave device.

A more detailed explanation of each of the class member functions that can be called as code modules from TestStand can be found in *Annex H - EthernetControl Class Reference*

12.2 Control software GUI for demonstration and debugging purposes

The main objective of equipment similar to the HDMI generator prototype is to use it in production, so the class library to interact with the test equipment from TestStand is the basic host PC software that must be developed. However, sometimes is very useful to have useful debugging software to interact with the equipment. For example in the HDMI generator board, the ADV7511 HDMI transmitter has a huge map of registers to play with, and although the direct I^2C commands allow direct access to low-level hardware, sometimes it is not easy to navigate through the map of registers without an appropriate tool with a user-friendly graphical interface (GUI).
There is also important to put the HDMI generator board in the correct context, as the board has been developed as a proof-of-concept prototype of a new hardware architecture for low-cost test equipment, and for demonstration purposes, sometimes is more useful to have a control application with a graphical interface than a set of libraries for test automation.
These ones are the main reasons to develop an application with a graphical interface, the EthernetCode application that uses the same methods from the EthernetControl class library but provides a user-friendly way to interact with the HDMI generator board

The EthernetCode GUI has been subdivided in 4 zones as it can be seen in Figure 12.2.

- 1st block is used to set the connection parameters like IP address and port, and connect or disconnect with the Ethernet Board depending on the previous state.
- 2nd block is used to load some control plug-ins to interact with the HDMI generator board, for example to configure audio or video setup.
- 3rd block is used to monitor a small set of registers of the ADV7511 HDMI transmitter by using the direct I^2C access functions. The registers are load from a text configuration file with a defined format shown in Figure 12.1 using a new line for

every register and TAB characters as separators for register parameters like subaddress, read/write mask to show/modify only some bits of one-byte registers, etc...

```
// ------------------------------------------------------------------------
------------------------------------>>
// Name           ID     RW/RO Mask        Tab    Min    Max    Default
     OptionalComments
// ------------------------------------------------------------------------
------------------------------------>>
chip_revision     0x00   RO    0xff        1      0      255    0x10   Revision
of the chip
audio_regen_N     0x01   RW    0x0fffff    1      0      0x0fffff    0x00   N to
regenerate audio clk
adv7511_04        0x04   RO    0xff        1      0      255         0x00
```

Figure 12.1. Format of hardware registers file to load registers monitored in EthernetCode application

- 4th block is used as a simple text-mode terminal very similar to the Telnet client but with some useful options like command auto-completion and command history. It also have the extra benefit that the command is only send to the board when the user pushes Send or Enter key, and that recreates a real terminal, not like in the Telnet client where every character typed is sent to the board, usually generating command syntax errors when user tries to use it as a real terminal, for example trying to erase a typographic error with the backspace key, etc.

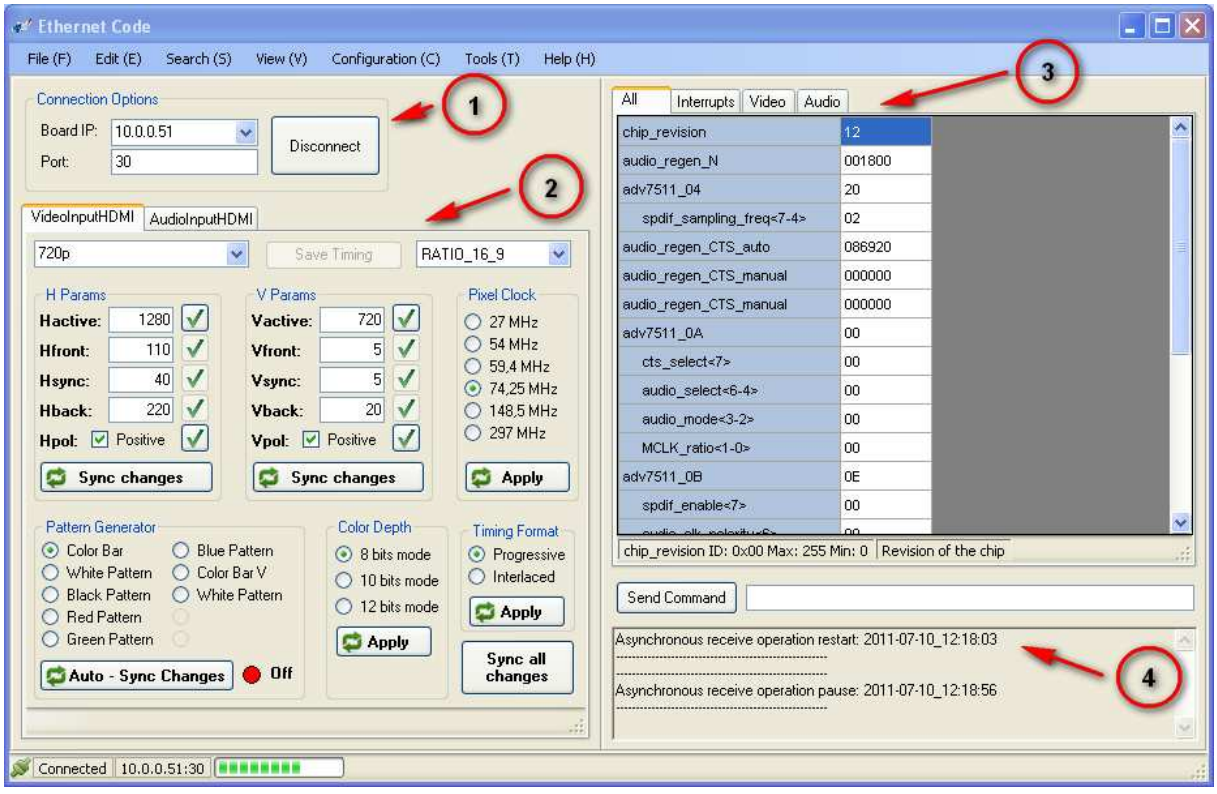


Figure 12.2. Ethernet Code application GUI, subdivided in 4 blocks

Chapter 13

Test sequence integration and tests in production condition

As it has been explained before during this work, the global CBA Low-Cost project consists not only of a new hardware architecture proposal but also a new mechanical and software architecture to reduce global costs of the automatic functional test systems.

As it has been explained in the previous chapter, the test sequences developed with TestStand offer the possibility to integrate code modules written in nearly any programming language. Apart from controlling the hardware needed for test and the DUT, test sequences created with TestStand are also in charge of test execution flow, reporting, database logging and connectivity to other enterprise systems for traceability.

The generic knowledge about the TestStand test management suite is outside the scope of this project, because it is more related to the new software architecture proposal instead of the new hardware architecture. Basically in this chapter it will only be briefly introduced the strictly necessary to understand the new hardware integration in the test sequence for functional test of LCD TV PCB boards.

Finally the chapter also explains the related tests made in production to assure the long term reliability of the HDMI generator board.

13.1 Introduction to Test Sequence for LCD TV main boards

In TestStand the basic skeletal structure to the entire test system is the process model.
A process model consists of a set of steps that are used to define the flow of execution of a TestStand system. The process model determines what actions to take before and after the test sequence executes

The current automatic functional test systems use the Sequential process model, the simplest model in TestStand. This model allows one Device Under Test (DUT) to be tested at a time and executes a continuous loop so DUTs can be tested one after the other.
Prior to testing each unit, the DUT is identified by reading its identification label. This methodology allows the common tasks repeated in the testing process to be separated from the tasks of a specific test of a DUT and that increases code reuse and flexibility while lowering development time.

The main test sequence is executed by the process model, and this is the sequence that really needs to be customized depending on the DUT. Another one of the main features of TestStand is its organization in the Setup, Main and Cleanup blocks. All the sequences and also the process model are organized this way, and that structure consists on that Setup block is executed first followed by Main block and finally the Cleanup block. If some error appears during the Setup, the Main can be skipped to directly proceed with the Cleanup, and in case of an error in the Main the normal process flow can be skipped to proceed directly with the Cleanup. This sequence flow allows for example to organize the hardware initialization

routines in the Setup, the real test of the DUT in the Main block, and finally release the hardware in the Cleanup.

On functional test systems for LCD TV main boards, CBAs, the sequence has several steps in subsequences to test all the functionalities of the TV that have been introduced in *Chapter 3.1: Introduction of LCD TV main boards testable features.* Usually several functionalities are grouped in steps, and that is the case for the HDMI related tests.

To know more about the organization of the test sequence, *Annex I: TestStand Functional Test Sequence architecture* contains the flowchart of the process model execution and also the main sequence execution. For more detailed information about TestStand process models and TestStand sequences development, some references are provided *[56][57][58]*.

13.2 Integration of HDMI generator board in Test Sequence

As the HDMI generator board is only used in the HDMI inspection steps, the explanation is focused on these steps.

First of all, it is needed to explain that the board has been testes in production by substituting the current solution in the CBAs, the commercial CVBS + Audio L/R to HDMI converter, explained in *Chapter 5.2.4 - HDMI generation commercial alternatives. State of* the art. The HDMI generator board has a direct added value in front of this solution that is the possibility to change the video formats and patterns, something that is required by TV design and that cannot be made in production with the currently used solution that generates a fixed 720p HDMI signal.

The Figure 13.1 shows the process flowchart of a normal HDMI input Test Step.

1. ESIIC HDMI multiplexer path is switched to put required HDMI signals from HDMI generator board in the correct HDMI connector under test.
2. Later the TV Board will be changed to correct HDMI Input using the serial debug protocol.
3. During the input change time, some extras will be checked to assure all signals of HDMI connector are correct.
   - HDMI EDID will be read by I^2C in order to check I^2C path from EDID to connector and correct data is already written in EDID
   - HPD signal from TV will be checked in ESIIC Board to assure signal path is correct.
   - CEC will be checked by assuring the signal path to uC. ESIIC will force transitions from 0 to 1, and the port in uC will be checked to see if transitions are detected. CEC software functionality is not checked, but it is no needed to assure hardware is ok.
4. After input signal is detected and with appropriate resolution, the video check and audio check will be made in parallel to reduce the test time. The video check consists on query the main microcontroller about the RGB values in some regions of the screen. The audio check consists in checking the audio output signal in the Data Acquisition board. In some HDMI inputs, audio will not be checked because signal path is already assured, in HDMI1 and HDMI4 audio is not checked, in HDMI2 audio is checked using Speakers output, and in HDMI3, audio is checked using Line Out output.

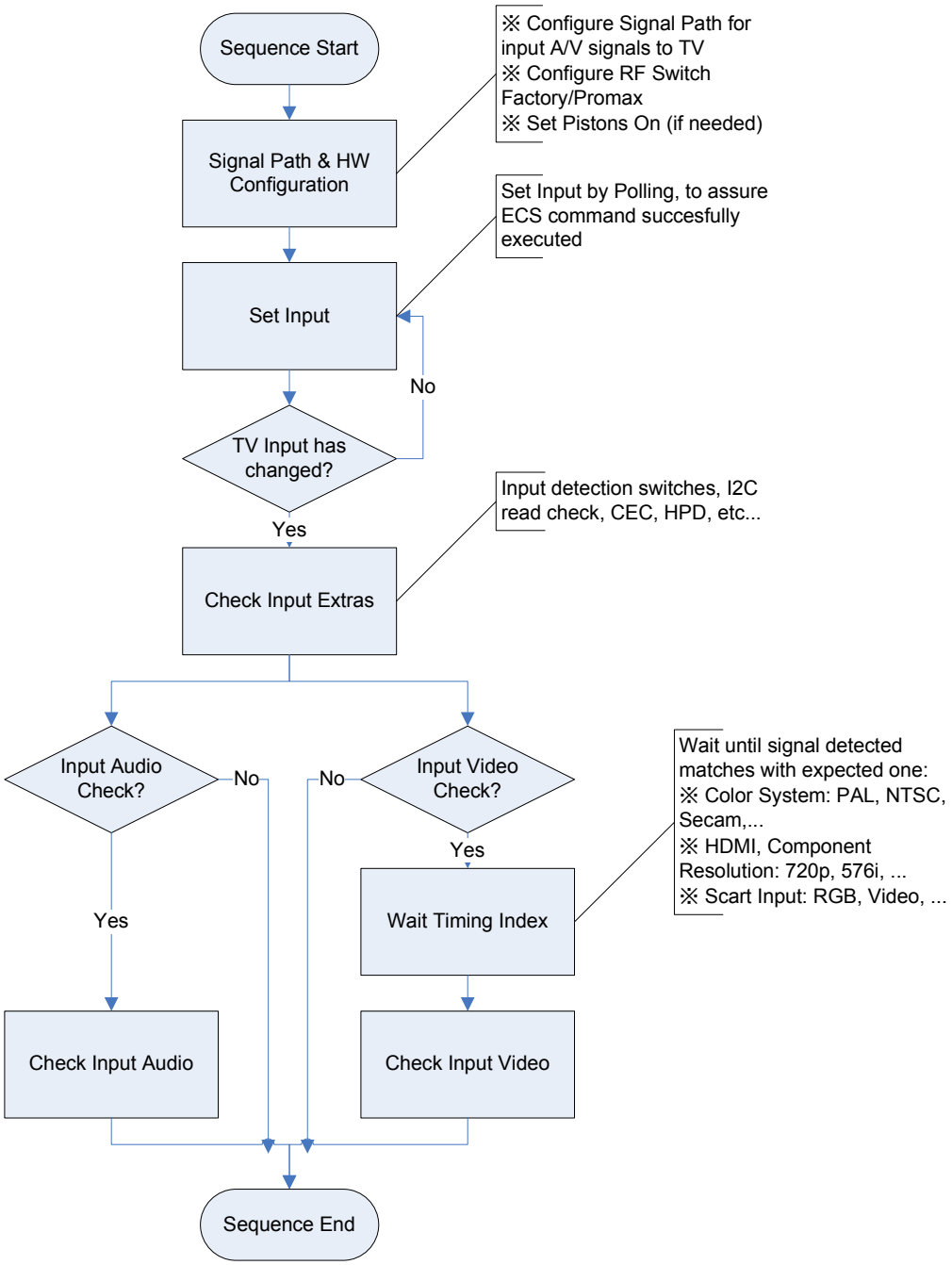


Figure 13.1. HDMI Input Test Step execution flowchart diagram

The finally adopted solution to take profit of the HDMI generator board extra features, is that different HDMI inputs will be tested at different resolutions, HDMI1 and HDMI4 will be tested at 720p, HDMI3 at 480p and HDMI2 at 1080p with 12bits deep color mode.

Before any test is made, the HDMI generator board is configured to the desired resolution in the WRK_HW_CONF.seq subsequence that is called from the main Test Step sequence. That sequence is responsible to configure the signal path from the equipment to every TV input.

The HDMI generator board configuration comprises three module calls:

- *ConnectWithEthBoard* method to create the connection with the HDMI generator board, called in Main block.
- *SetHdmiVideoSetupEthBoardTS* method to appropriately setup the video configuration, called in Main block after connection is established. Shown in Figure 13.3.
- *CloseConnectionWithEthBoard* method to close the active connection, called in Cleanup block.

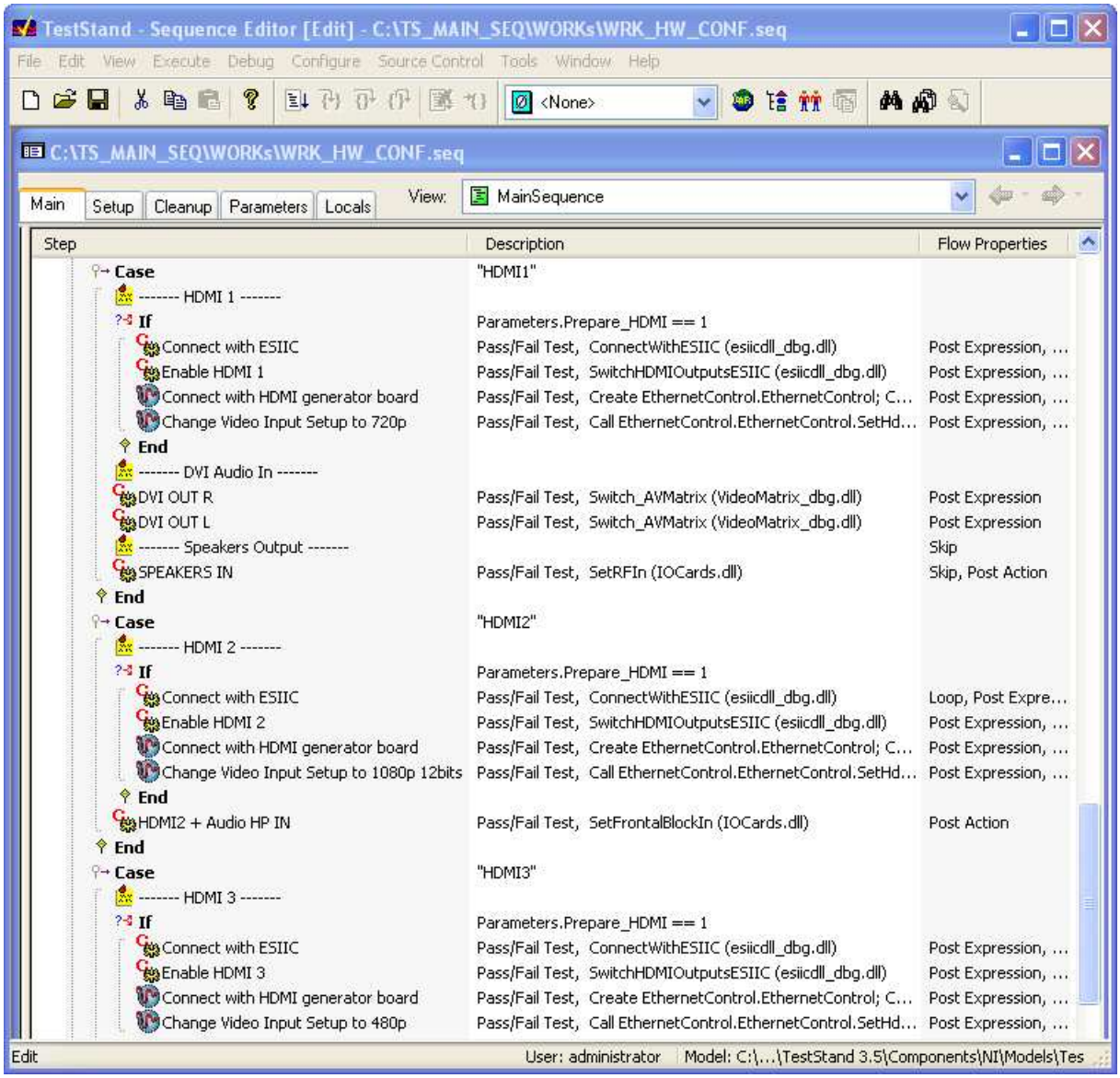


Figure 13.2. TestStand sequence WRK_HW_CONF to configure HDMI generator board

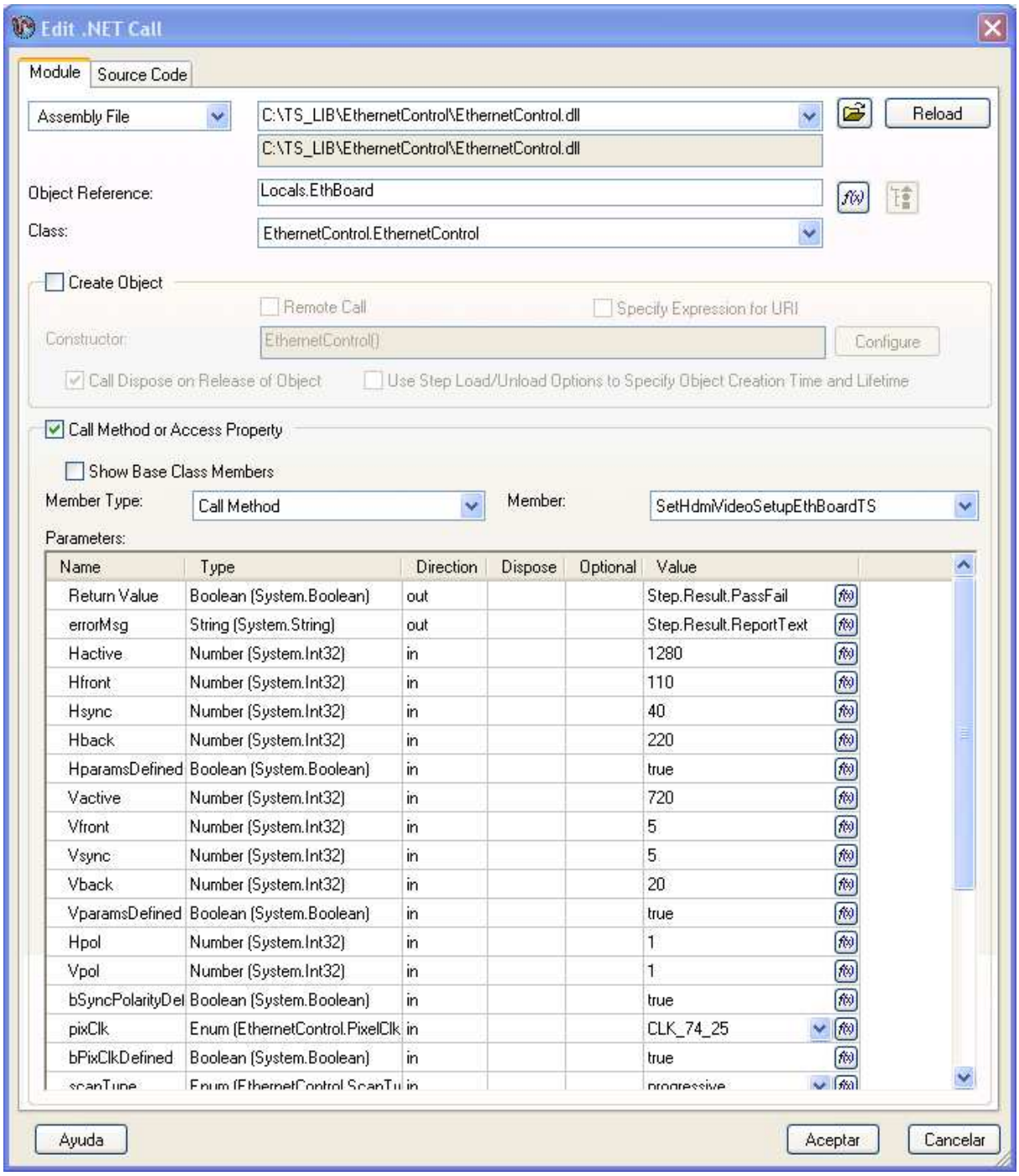


Figure 13.3. SetHdmiVideoSetupEthBoardTS method parameters configured from TestStand Edit Call dialog

13.3 Production tests incidences and results

The HDMI generator board was mounted in the current CBA and all modifications in test sequence were implemented in order to configure the board in every HDMI Test Step.
The first incidence was the coexistence in the same network with the ESIIC board that is also controlled with a similar ethernet remote control interface and that used the same IP address, 10.0.0.51, than the HDMI generator board, that finally was changed to another default IP address.

The next problem appeared with the default behavior of the HDMI generator board that remains in low-power mode until an HDMI connection is established. Although the board is permanently connected to the ESIIC board HDMI multiplexer input, some of the ESIIC boards don not have a correctly written EDID because with the current HDMI generation hardware it was not needed, as the commercial CVBS + Audio L/R to HDMI converter was always generating a fixed HDMI 720p signal. Although the CBA selected had an ESIIC board with a correct compliant EDID, it was decided to also implement in the HDMI generator board software, a special mode to force the power-up without necessity to read a correct EDID from the sink. This mode is useful to force an HDMI signal without taking care about the condition of the HDMI sink, and eventually it was decided to use this mode during the production tests.

Finally some specification adjustments were needed, to adapt the values read from TV regions during the viewless automatic video tests. That means that the read RGB values were slightly different from the ones read when the commercial CVBS + Audio L/R to HDMI converter was in charge of the HDMI generation.

The results during the production tests were satisfactory, as it was not noticed any increase of incidences or higher NFF ratio in HDMI related Test Steps. Also the coverage improved because the HDMI receiver in the main microcontroller of the TV board and the internal HDMI switch were tested at different resolutions instead of a fixed one, increasing the detection ratio of defective components.

Anyway it was noticed that some of the HDMI generator board features were not fully used, because the ESIIC multiplexer in the middle of the signal path between HDMI generator board and the TV, was still in charge of many of the HDMI tests, for example the EDID read test, CEC tests and HPD tests. The new hardware architecture proposal was based on putting the smart test equipment close to the DUT, basically directly connect it to the DUT without any interconnection interfaces in the middle. So to test properly the HDMI generator board as the proof-of-concept prototype of this new architecture, the HDMI generator board should have been directly connected to the LCD TV main board.

Chapter 14

Conclusions

The purpose of this chapter is to give an overview of the whole project, the results and goals achieved and make a brief outline of the future lines of work.

14.1 Project accomplishments summary

After finishing the research work introducing the successful results of the proof-of-concept prototype development, we can realize that the main objectives of this project have been accomplished.

Competitive solution in the test & measurement market for much lower cost
Firstly, it is important to remark that the alignment of the project with the top trends identified in test & measurement industry has been demonstrated during all the work.

- Increased use of parallel test systems. The new hardware architecture, focused on individual test equipment for each DUT, enables the development of automated test applications capable to achieve the highest possible throughput in production.
- Growth of software-defined instrumentation. Most stand-alone instruments cannot change their functionality as fast as the market addressed evolves due to a fixed firmware and an approach not focused on flexibility. The new hardware architecture focuses on the flexibility of the test instrumentation to address the obsolescence problem of the instruments in the long-term.
- Growing popularity of FPGA-enabled instrumentation. The embedded system architecture selected to implement the new hardware architecture proposal is aligned with the new trends of the high-end instrumentation market, but for a much lower cost.

All these trends in the industry have also been identified for the major agents in the market, which offer solutions also aligned with the evolution in the industry, but only destined to the high-end segment of their test & measurement products. For example, National Instruments offers the PXI platform as the solution for systems that require high-performance and modularity. PXI systems are an example of a widely used software-defined instrumentation standard for building modular, reconfigurable high-performance automated test systems. National Instruments also offer some special PXI modules like the reconfigurable I/O (RIO) FPGA-based hardware, to give some solutions to the demand of reconfigurable test hardware.

In *Chapter 3: New Hardware Architecture for Functional Test of LCD TV main boards*, it has been introduced the hardware architecture of the current functional test systems addressed to LCD TV main boards, which is based on the classic monolithic approach of commercial stand-alone hardware necessary to test the DUT. The new hardware architecture proposal on the contrary is thought to fight the monolithic idea of the accumulation of different test equipments for every different electronic product, by presenting the “platform” approach, that focuses on the design of an open and reconfigurable platform that would allow the engineering team to give quick solutions for any new test necessity in the classic product, or new electronic products.

This is the same idea behind the National Instruments PXI systems but the main problem of all these solutions, is that by nature they are focused to the high-end segment of the test & measurement industry and the prices for basic PXI platforms are really expensive.

In conclusion, that "platform" approach is one of the most ambitious goals of this project, and the solution proposed really shares this philosophy with the state-of-the-art test systems like PXI, both competing with the classic test hardware approach. But in front of PXI, and the classic accumulation of test equipment, used in current functional test systems at BCN Tec, it offers a much lower cost and a better adaptation to the final functional test system, because the design is already designed from the start to work in a very specific kind of test system.

Another important aspect besides the typical project gains and savings is the added value that cannot be converted into a money account, and in this case the know-how achieved by running this project is possible to apply to any other specific cases in the future. Basically it allows the engineering team to be prepared for any challenges in the foreseeable future,

Proof-of-concept prototype success on showing the feasibility of new hardware architecture

One of the main objectives behind the development of the proof-of-concept prototype was the possibility to compare the goodness of the solution against others in the market, and this was one of the reasons to target a relatively new technology like the HDMI.
Although it is important to not lose the focus of the global solution proposed, the HDMI generator board is successful in any raw comparison with other commercial options, and again comparing with a platform solution with PXI, the price differential is huge, as the approximate cost of PXI modules for digital video generation is around 18k€.

Another important aspect where the proof-of-concept prototype can be viewed as a success is on showing the feasibility of the main concepts behind the original idea. The possibilities of the "platform" approach has been demonstrated when all the target features of the HDMI generator board has been accomplished. Besides, none of the possible future update has been compromised by the design, as the hardware and software have been developed with the scalability in mind.

It is important to remark here that the proof-of-concept prototype cannot be viewed as a specific solution for HDMI generation. The correct way to look at it is how straightforward has been to introduce HDMI generation given a functional and scalable platform. So, although the HDMI generator board keeps its own directly compared with specific solutions for HDMI generation in both terms of cost and features, really the proof-of-concept prototype development results has to be evaluated in two parts.

- The creation of a structured, scalable and reconfigurable platform to achieve different advanced test capabilities.
- The process to add and important capability like HDMI generation to the platform.

Cost cutting is one key strategy adopted by almost all businesses to sail through the current poor economic conditions, and to be a creator of test equipment more than an integrator of test equipment is a good cost cutting strategy if the development costs can be kept at the minimum. The project demonstrates that given a powerful configurable platform and a good design methodology, the development costs to add any new challenging technology can be deterministic. This is really important if the company does not have the possibility to throw the chequebook for commercial equipment any time a new product is introduced in

production. It is still more critically important in companies trying to compete in the CMD business, where potentially electronic products from very different sectors can be introduced in the factory. In conclusion, the project is successful in proposing a viable low-cost alternative to the commercial equipment approach, and this result is still more important in current economic context, with investments kept at the minimum.

Proof-of-concept prototype reconfigurable platform success
The conclusions of the different development phases of the platform can be summarized as:

- From hardware point of view, the main conclusion is that apart of the core generic capabilities of the platform, that should address generic test necessities like data acquisition, DVM measurements, etc., the final embedded system has to be expandable through expansion connectors for specific functionalities like HDMI generation, that is focused only to TV boards.
- From system and reconfigurable hardware point of view, it has been developed a scalable system ready to integrate and reuse any cores from any given source, usually Opencores or homemade cores with a wishbone interface. The success has been demonstrated by integrating all the needed cores for HDMI generation and easily interacting with them from the main controller, NiosII.
- From embedded software point of view, it has been developed a multitasking system based on uC/OS-II real time kernel, ready to continuously integrate the control of any new extra capabilities implemented in hardware or external peripherals. It also has been introduce a generic ethernet control interface that can be used to develop easily other task commands for new functionalities in the basic platform. Again, the integration of HDMI-related tasks in a system ready to run any possible tasks it has been a success. In fact the system is clearly under-used with only the HDMI generation tasks implemented.

Proof-of-concept prototype success integrating a challenging technology: HDMI
The conclusion taken during the integration of a challenging technology like HDMI, which is still viewed as a premium capability paid at premium cost in most of the current commercial test equipment, can be summarized as:

- Features implemented in the HDMI generator board makes it competitive directly compared with other commercial solutions, so it demonstrates that the new hardware architecture not only can give solutions for straightforward test necessities like data acquisition or DVM measurements, but with the appropriate scalable platform it also can challenge the premium features. In fact, the HDMI transmitter solution taken from HDMI generation and the scalability of the platform allows to continuously update it to target other HDMI-related features left out of the scope in this initial phase, like CEC, HDCP, etc.
- All the targeted features of the HDMI generator have been successfully accomplished, and it have been proven during the power up and validation phases of the design.
- A generic control library for any embedded system with an ethernet remote control interface, ready to accept commands in an open socket connection, have been also successfully developed.

14.2 Next development phases of the CBA low-cost project

As this project is intended as the introduction of new hardware architecture for functional test systems, the future work depends directly on the success of this initial phase.

The success of the proof-of-concept prototype allowed the project to go beyond the first major milestone. It was decided to develop the first prototype of the Ethernet Acquisition Board presented in *Chapter 3.2: New Hardware Architecture Proposal* that is targeting some generic features like:

- High-speed signal acquisition (DAQ)
- Analog Audio Analysis
- SPDIF Digital Audio Analysis
- LVDS DesSerialization & Capture
- Digital Video Capture (HDMI receiver)
- Analog Video Capture
- Voltage/Current Measurements
- Programmable Inputs (PIO) for basic low speed logic analyzer functions

In contrast to the proof-of-concept prototype, that have been designed from a known platform like the 2C35 development kit, this first product prototype have been started from scratch, it means that it also adds the hardware development of the system as the main challenge, something that have been left outside the scope of this initial project.

The final CBA low-cost test system, comprising also a new software and mechanical concept, have been left on standby because of the current critical economic situation, that completely cuts any development if it cannot be assigned to a specific product introduced in the plant, that really needs that test potential and justifies the development costs. Anyway the success of the initial phase of the project allows the possibility to continue the project if the current situation on the plant evolve to a more favorable scenario.

14.3 Future lines of work and research

In the specific field of the HDMI generation some of the most new features have been left unexplored and to add them could be seriously considered if the work in the HDMI generation would have to be continued.

- Support for different color spaces and color component subsampling like YCbCr 4:4:4 or YCbCr 4:2:2. Limited to RGB 4:4:4, color space converter no used.
- HDCP digital content protection. Video and audio are always sent unencrypted in the current solution.
- ARC audio return channel analysis. It could be implemented in the future with the same HDMI transmitter solution integrated in the final embedded system platform.
- Generation of special 3D video formats. It could be implemented in the future with the same HDMI transmitter solution integrated in the final embedded system platform.
- CEC controller functions. It could be implemented in the future with the same HDMI transmitter solution integrated in the final embedded system platform.

In terms of the platform development, the next project should consider to use more efficiently the board resources and potential, like all the dynamic memory and flash storage. For example in the proof-of-concept prototype, the platform would have allowed to generate more advanced test patterns from image files stored in flash memory. The Ethernet interface that it is only used as a remote control interface could also be used to stream contents to/from the Host PC. During the embedded architecture selection the high-performance potential of the FPGA+Soft-core alternative was one of the main reasons behind this selection, and in the

proof-of-concept prototype that potential has not been fully exploited as the focus was on showing the scalability potential more than the performance potential to implement high-end features.

Finally in terms of software development, more investigation is needed on the inefficiency while serving interrupt requests of the current combination of NiosII uC and the uC/OS-II kernel. The investigation should be centered first on changing the NiosII uC options selected in the SOPC Builder tool, because the current configuration was thought for the debug phase and any optimization is still applied. From software point of view uC/OS-II should also be compiled in release version applying some optimizations.

Finally, a new real-time kernel better prepared to achieve the flexibility objectives could also be tested. uC/OS-II theoretically offers a very fast task switching and interrupt handling, but other systems like uClinux for example, offer a complete filesystem, TCP/IP and video functionalities integrated in the kernel, and ability to quickly add other peripherals to the system. Potentially it could provide a better platform to use it as the kernel of the main platform instead of uC/OS-II because of the ease of use and a lot of applications and literature available about it.

Annex A

Mount process with SMT and HM for TV PCBs

In the following process flowcharts it can be seen how a two-sided TV main board is produced from the blank boards to the finished goods, that later are sent to the TV set assembly lines, and after, it is clearly pointed out which processes, inspections and tests are made to assure a proper quality. The process is subdivided in two blocks, Surface Mount Technology (SMT) and Hand Mount (HM).

A.1 Surface Mount Technology (SMT) Area

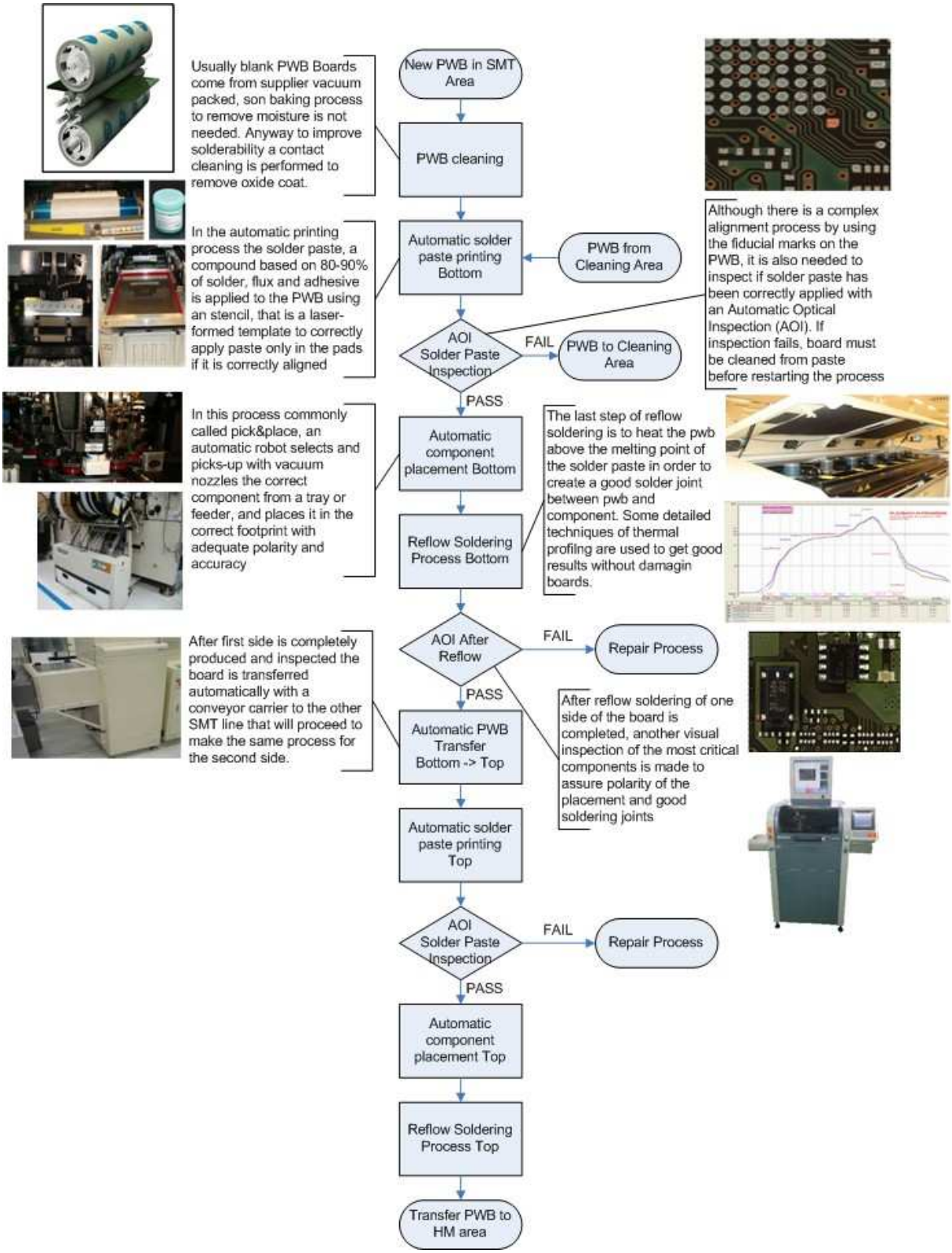


Figure Annex A.1. SMT process flowchart

A.2 Hand Mount (HM) Area

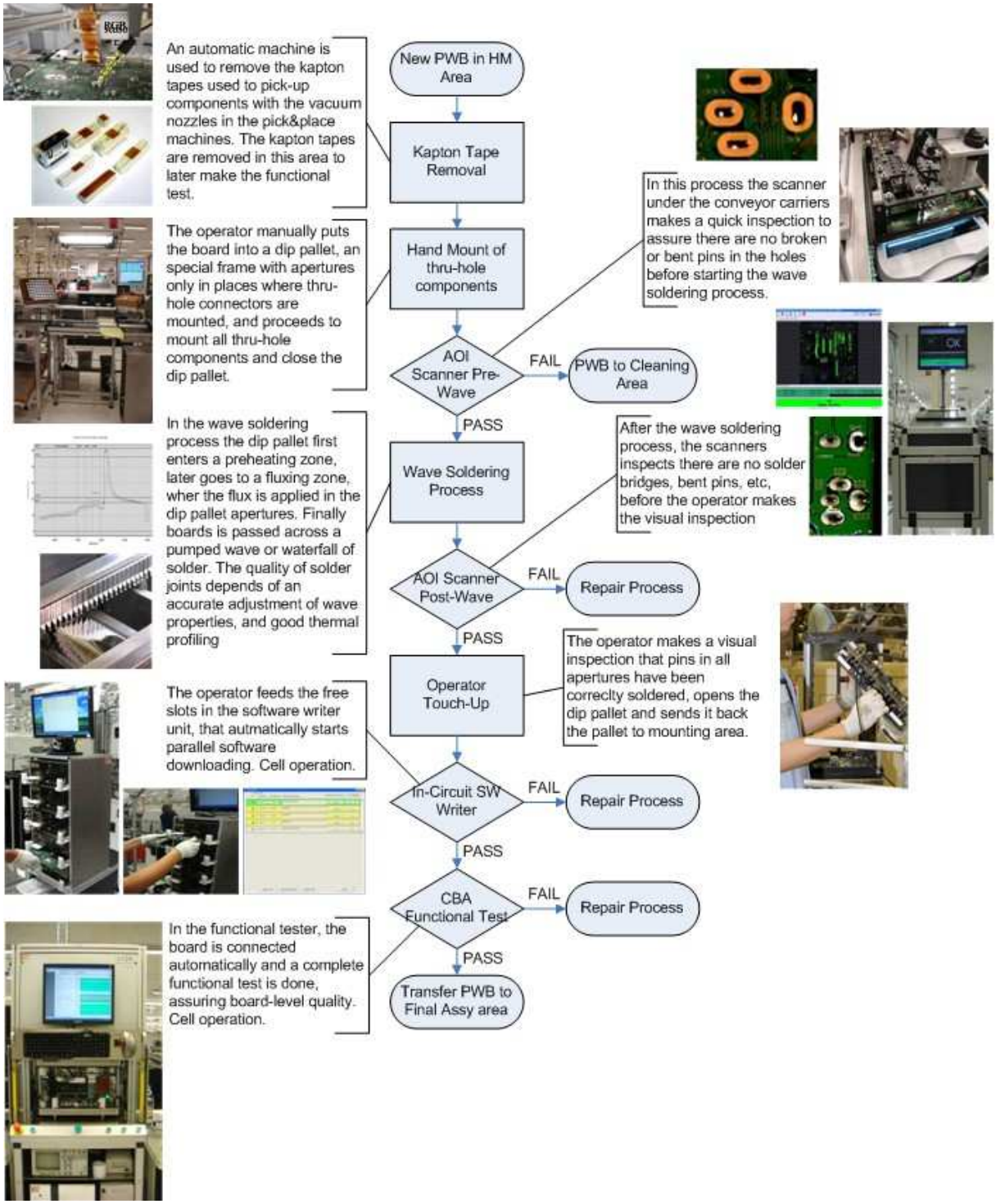


Figure Annex A.2. HM process flowchart

Annex B

Embedded System Design Methodology

B.1 Introduction to embedded design for soft-core based systems

The following sections describe in a generic way the Altera design flow for embedded systems based on NiosII soft-core processor, which basically consists in three phases of development:

- Hardware design steps
- Software design steps
- System design steps involving both hardware and software.

That embedded system design flow is summarized in the following Figure Annex B.1, where the system, hardware and software design steps can be clearly identified, as well as the relationships and different triggers to start the development of every phase. In all the different design phases, Altera tools for system, hardware and software development will be progressively introduced, as well as the fundamentals to understand the design methodology to create an NiosII based system platform.

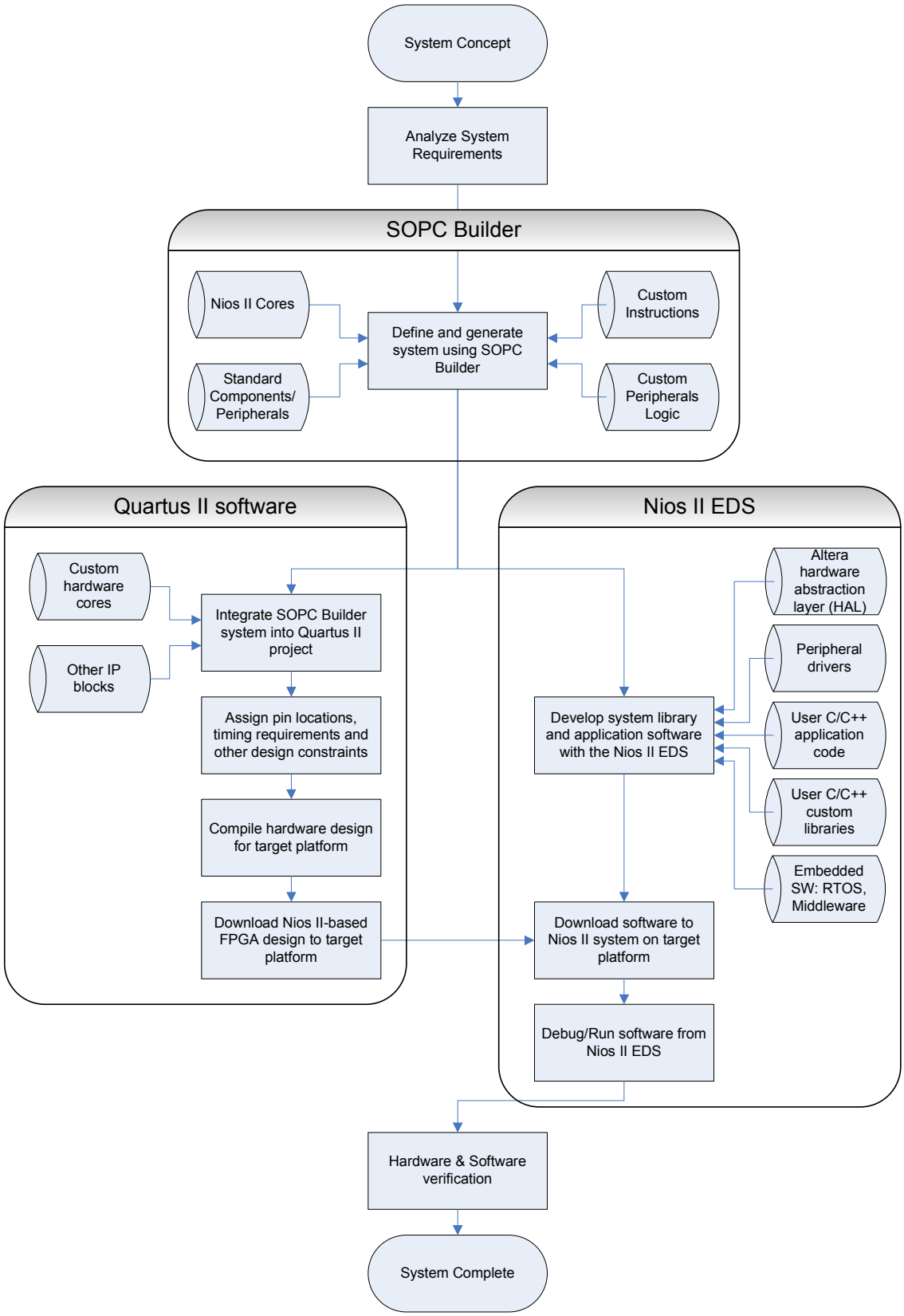


Figure Annex B.1. Typical flow for embedded system design using in a NiosII system

B.2 System Design Flow

B.2.1 Introduction to SOPC Builder

The system definition is specified using SOPC Builder tool. As PLD complexity has increased, the system design methodology has changed in order to accelerate designs and reduce time-to-market. SOPC builder main objective is to simplify this task with a friendly block-based design and a higher level of abstraction, which makes easier the task to integrate hardware modules from different sources on a single chip, and interface with other complex modules or processors inside or external to the FPGA.

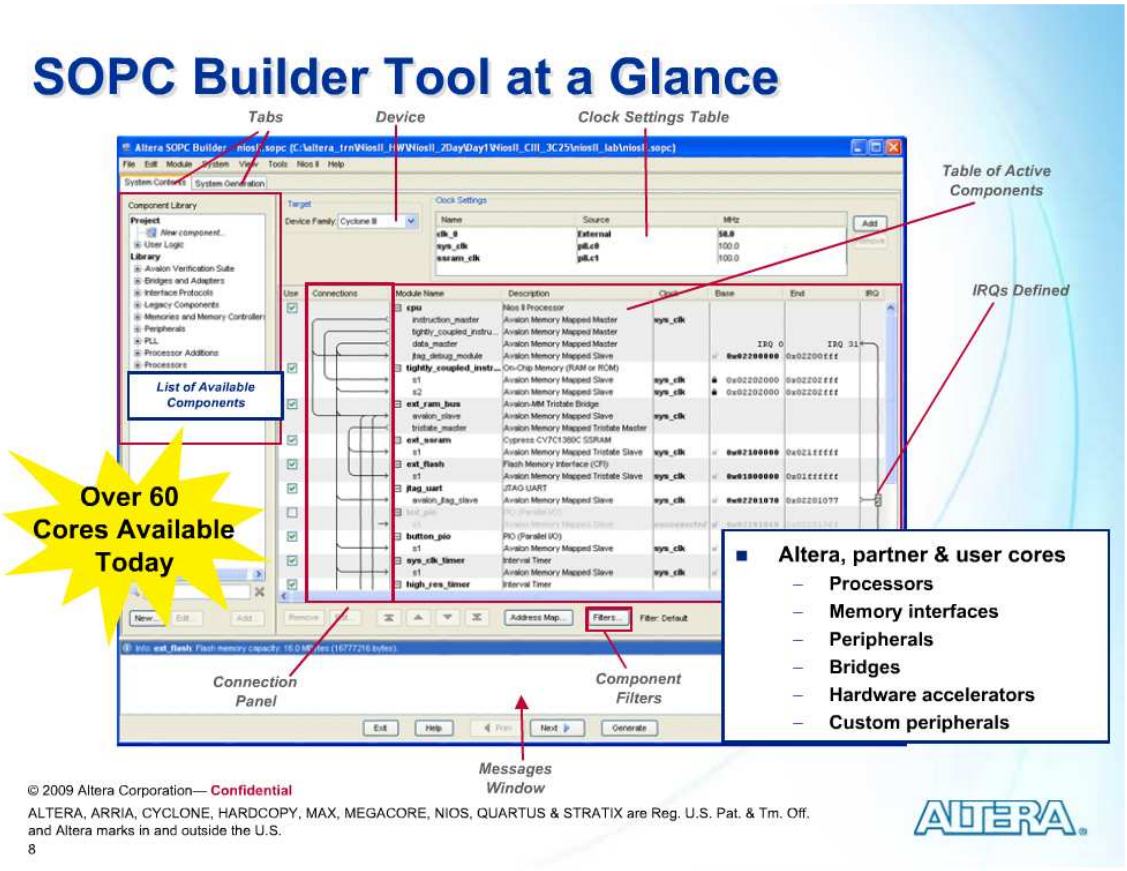


Figure Annex B.2. SOPC Builder main graphical interface (GUI)

SOPC builder is a powerful tool, but a complete explanation of their capabilities is not the main purpose of this project as there is a large amount of documentation available, so the short explanation will be focused on how to correctly plan the system design using this tool.

B.2.2 System Design with SOPC Builder and Core Platform System Design

After making a pre-design activity oriented to discover the main system components necessary to fulfill the design requirements, the system definition can be summarized in three steps.

1. Select components.
2. Make connections between components
3. Generate system.

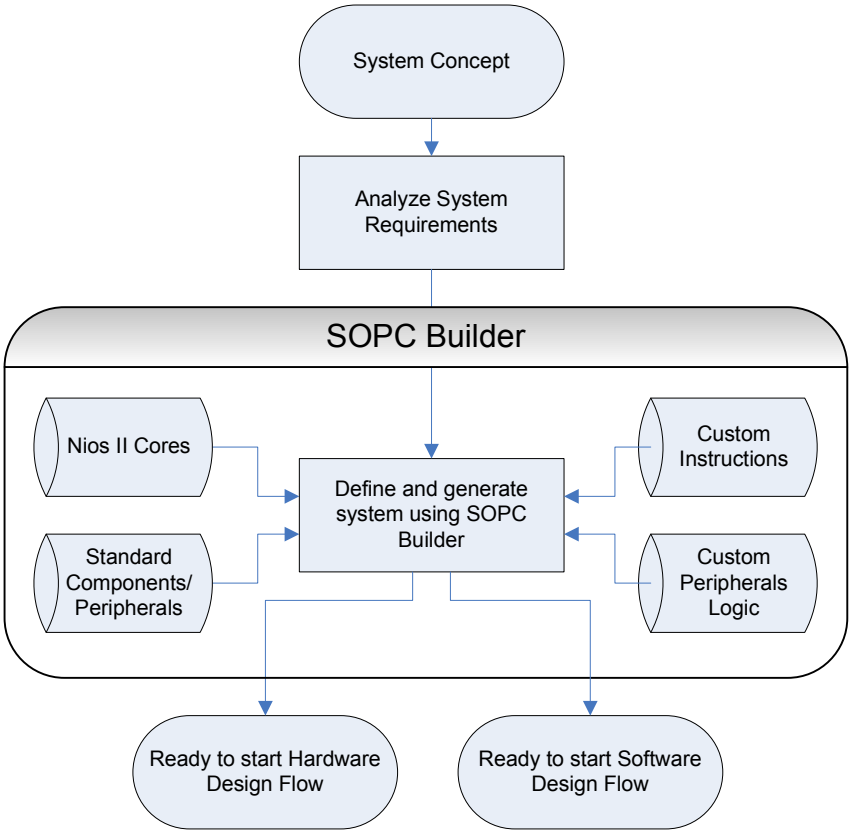


Figure Annex B.3. System design flow

Selecting the components

After analyzing the system requirements, SOPC Builder is used to select between a large list of modules, called components, that are the building blocks for creating the system. Every module use the proprietary Altera logic bus, called Avalon to communicate with the other modules.

The specification defines a lot of different Avalon standardized interfaces to use in predefined or custom components:

- Avalon Memory Mapped Interface (Avalon-MM), that is the typical one and more commonly used in designs, consists on an address-based read/write interface used for master/slave connections.
- Avalon Streaming Interface (Avalon-ST), an interface created for big unidirectional flows of data between a source and a sink.
- Avalon Memory Mapped Tristate Interface, that is very similar to Avalon-MM, but used to support off-chip peripherals by sharing data and address buses and reducing FPGA pin count.
- Avalon Clock, an interface that drives or receives clock and reset signal to synchronize interfaces and provide reset connectivity.
- Avalon Interrupt, an interface that allows components to signal events to other components.
- Avalon Conduit, an interface that allows signals to be exported out of the system level design in order to be connected to other on-chip or off-chip modules.

Although it is a proprietary bus, Avalon bus has an intentional ambiguity in the specification of their interfaces, as not all the signals must exist in the slave-side in order to complete a correct transfer. There is a large amount of information about Avalon specification, and to

explain it is not the purpose of this project, so basically the main idea that has to be clear, is that modules communicate with each others with Avalon interfaces and SOPC builder will be used to connect any logical device (either on-chip or off-chip) that has an Avalon interface, and every single component can include any number of these interfaces.

The components included in the final system can be selected from standard Altera components, third-party IP blocks provided as SOPC Builder-ready components and custom homemade IP blocks converted using SOPC Builder component editor.
The list of selectable components can be summarized in several categories. The references in this project can be used to have a more extended view of the current SOPC Builder-ready cores available.

- NiosII Processors. When NiosII component is selected, the processor core type, level of cache, debugging options and custom functionality can be configured to completely adapt the selected processor to design required functionalities.
- Memories and Memory controllers. To implement on-chip fast memories, DMA master engines or high-performance controllers to access off-chip resources, like SRAM, SDRAM, CFI Flash, SSRAM, DDR SDRAM
- Bridges and Adapters. For high performance data flow between slave interfaces to Avalon-MM Master interfaces.
- Communication interfaces. Support for high-performance communication interfaces with a lot of different protocols, like Ethernet controllers, PCI, PCIe, SPI, Serial communications, etc.
- System Peripherals for NiosII processors, like JTAG Uart, System ID, Timers, Programmable I/O, PLL, etc.
- DSP IP cores. Cores for video and image processing, filters, etc.
- Legacy Components.
- Custom Peripherals.

SOPC Builder uses the term “component” to describe all kind of hardware modules included in the system. Anyway, in the following sections, “component” is used interchangeably with “device” and “peripheral” when the context is closely related to SOPC Builder.

Also an important feature of SOPC Builder is the options available to find the best balance of hardware and software resource usage, when there is some software bottleneck in the final application. There are some acceleration methodologies to implement relevant algorithms in hardware instead of software.

- Custom peripherals. For example DMA engines to offload CPU resources.
- Custom instructions. Hardware coprocessors that can be used in parallel to main CPU unit for critical algorithms.

The basic components selected to implement the basic platform system that has been introduced as a proposal in the previous chapter are summarized in the block diagram in Figure Annex B.4.

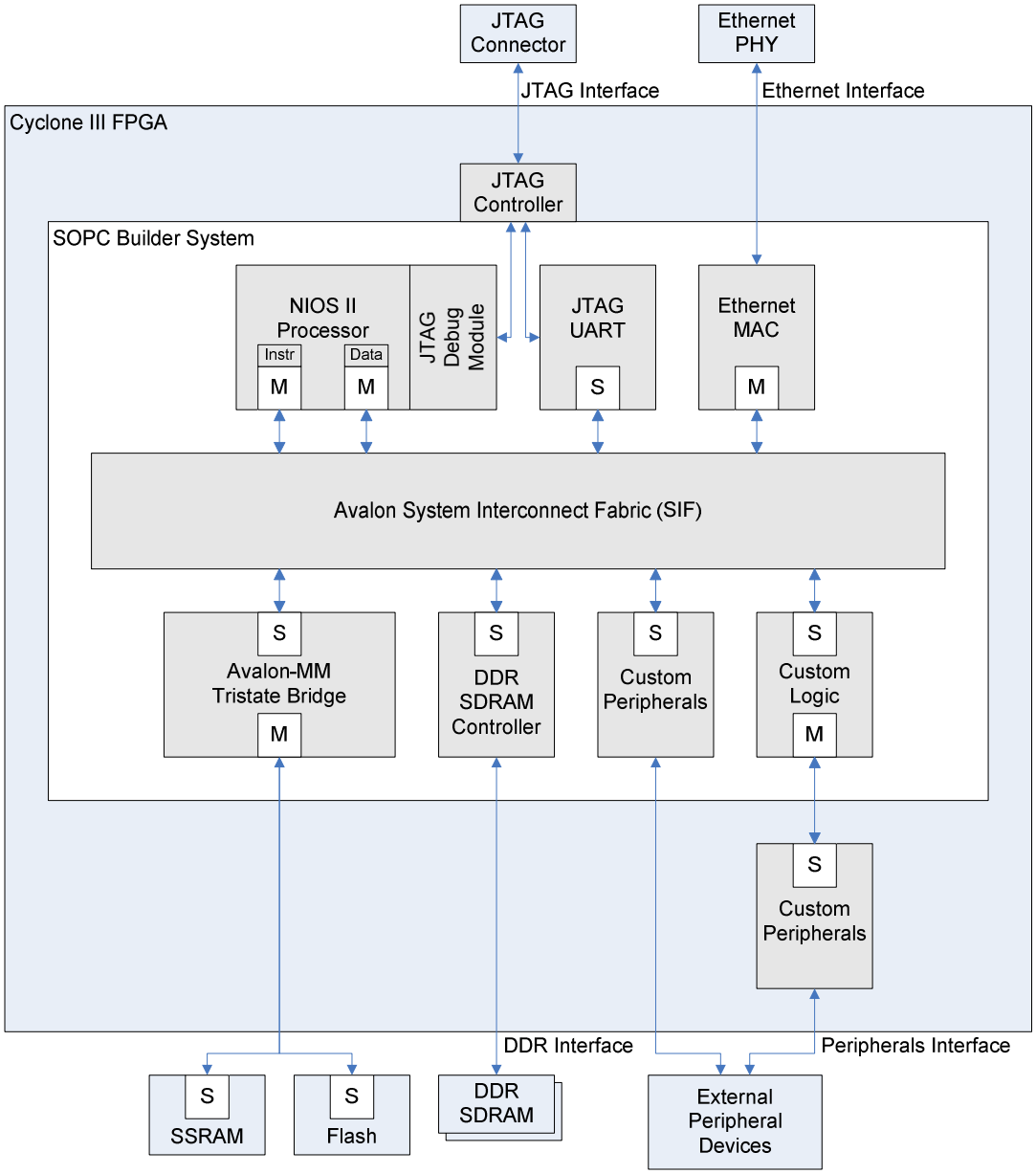


Figure Annex B.4. Block diagram view of the NiosII system designed with SOPC Builder

Making the connections between components

When all the components included in the final system have been selected and individually configured with the desired options, the control plane must be created by interconnecting each of the different master interfaces to all the slave interfaces of the selected components that master needs to access. SOPC Builder tool makes easy this process by graphically making in a cross-matrix window each one of the interconnections. SOPC Builder generates automatically the System Interconnect Fabric (SIF) that is the collection of interconnect and logic resources that connect Avalon-MM master and slaves on components in a system. It guarantees that signals are routed correctly between master and slaves, as long as the ports adhere to the rules of the Avalon Interface Specification.

In the next Figure Annex B.5, there is a simplified inside vision of the SIF in a memory-mapped system with multiple masters. In that system, all the masters have the ability to access to every memory interface and all the multiple masters can be active at the same time, simultaneously transferring data with independent slaves.

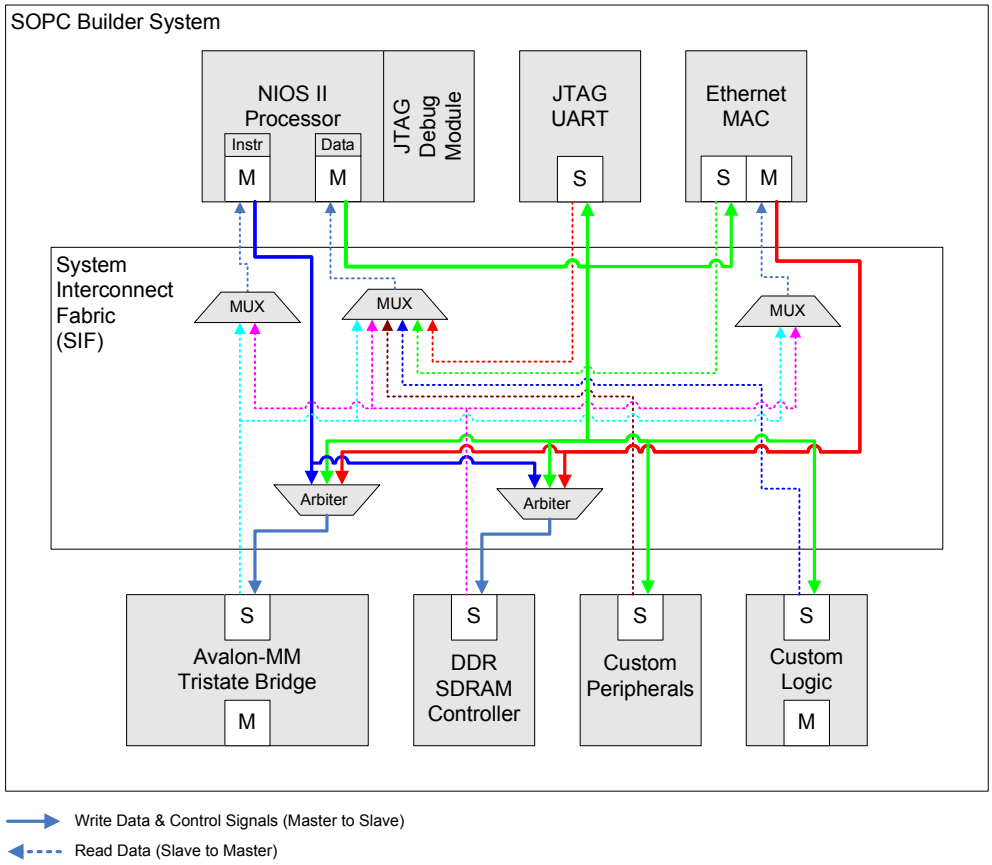
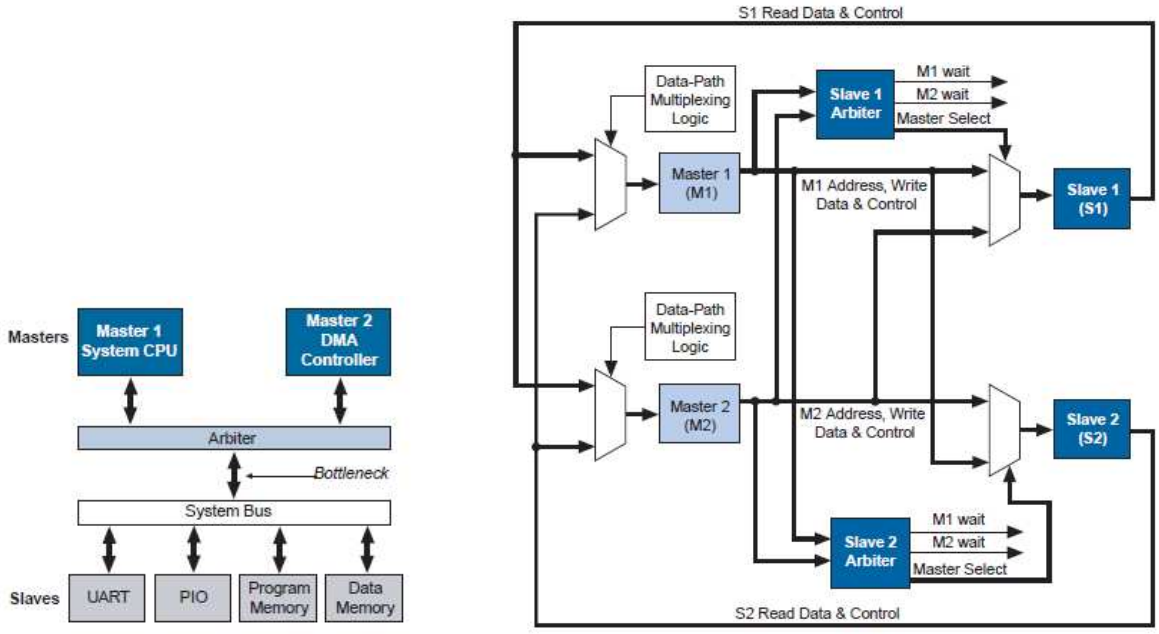


Figure Annex B.5. Detailed view of the automatically generated SIF in the NiosII system

This architecture known as slave-side arbitration eliminates the bottleneck when accessing to a shared bus, which is typically present in other microprocessor systems that can not take profit of the natural parallel hardware in a FPGA structure. This parallel architecture can be used to create concurrency as several bus transfers and computational processes can occur at the same time. The usage of multiple masters is a common technique in system design with SOPC Builder because it exploits the natural features of the FPGA architectures. In the following Figure Annex B.6 it can be graphically appreciated the differences between these two architectures in terms of efficiency. The main drawback that is the complexity of the slave-side arbitration architectures is solved by the fact that SOPC Builder tool generates these interconnections automatically, dealing with all the complexity and allowing the user to concentrate on another issues.



(a) Shared bus arbitration typical in other microprocessor systems.

(b) Slave-side arbitration in a NiosII system automatically generated with SOPC Builder.

Figure Annex B.6. Shared bus arbitration vs slave-side arbitration

As a summary SIF logic provides the following functions and SOPC Builder automatically takes care of them after configuring the components properties and making the appropriate connections between them:

- Address Decoding. Configured by assigning base addresses in the list of active components on the System Contents tab.
- Datapath Multiplexing. The multiplexing logic is specified using the connections panel on the System Contents tab.
- Wait State Insertion. SOPC Builder generates wait state insertion logic based on the properties of all slaves in the system.
- Pipelined Read Transfers. SOPC Builder generates pipeline management logic based on the properties of the master and slaves in the system.
- Dynamic Bus Sizing and Native Address Alignment.
- Arbitration for Multimaster Systems, based on slave-side arbitration instead of shared-bus arbitration.
- Burst Adapters
- Interrupts
- Reset Distribution

Generating the system

After adding components to the system, configuring all of them and specifying connectivity, the system is ready to be generated. During the system generation, SOPC Builder creates two kinds of different outputs in order to continue with hardware and software development:

- The HDL files that the Quartus II software compiles to generate the configuration file for the FPGA. Theses files are needed to start with the hardware design flow.
- The SOPC Information File (.sopcinfo) that contains a system description that the software development tools like the NiosII EDS use to generate a system library,

called Board Support Package (BSP), specific to that SOPC Builder system. These files are needed to start with the software design flow.

B.3 HW Design Flow

B.3.1 Introduction to Quartus II

After generating the system with SOPC Builder application, the system needs to be integrated to a hardware project by using Quartus II software. Quartus II is a software tool for analysis and synthesis of HDL designs, and provides all that developer needs to compile designs, perform timing analysis, simulated designs, etc. Anyway, as explaining Quartus II capabilities is not the goal of this project, the HW design chapter will be focused in how SOPC Builder interacts with Quartus II.

B.3.2 Hardware Design with Quartus II

When the system design phase is completed, the HDL files generated with SOPC Builder are instantiated and connected to other FPGA logic outside of the NiosII System with Quartus II software. The hardware design flow can be summarized in four steps.

1. Instantiate SOPC Builder system from HDL files and connect signals from system to other signals in the FPGA logic. In this phase some other custom hardware modules or IP cores from third part providers, which are not ready to be directly integrated in the system with SOPC Builder tool, can be connected to the system.
2. Connect signals from system or a different FPGA logic to the external components or peripherals present in the target board. This is the final integration with board level design, as specific pin locations for I/O signals must be assigned here.
3. Constrain the design to meet timing or other requirements using tools provided by Quartus II software. When the final FPGA-based design is constrained, it must be compiled for the specific device in the target board.
4. Download design to target platform. When compiling design, Quartus II software generates the programming files that will be used to download the Nios II-based FPGA design to the target board and configure it. After configuration, the FPGA behaves as specified by the hardware design, and now the platform is ready to download software and run applications created for this specific configuration based around a NiosII processor.

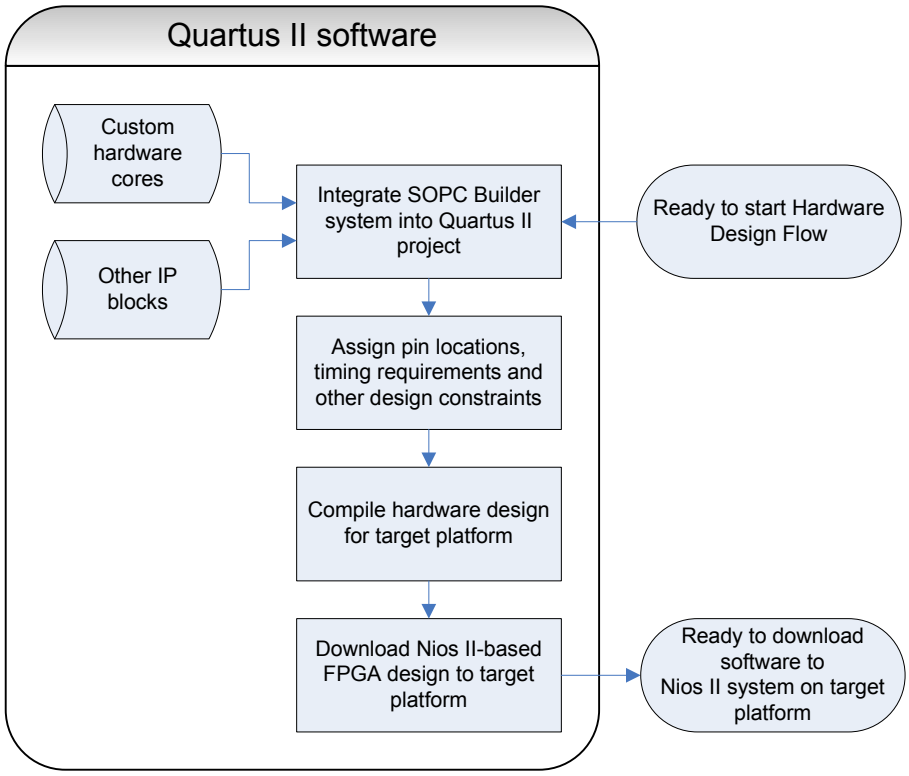


Figure Annex B.7. Hardware design flow

B.4 SW Design Flow

B.4.1 Introduction to NiosII Embedded Design Suite

After finishing the system design flow, the output files of SOPC Builder are used to start the software design flow. In addition, the hardware design flow must be finished to have a fully working target platform, where the software that will be created for that specific NiosII system will run.

All the software development tasks for the NiosII processor system can be done with the NiosII Embedded Design Suite (EDS) software development tool. The NiosII EDS provides a consistent software development environment that works for all NiosII processor systems and includes proprietary and open-source tools (such as the GNU C/C++ tool chain) for creating NiosII applications. The software designer can select from two different programming environments.

- NiosII Software Build Tools (SBT) Command Line. A complete console environment for typical command line development flow oriented to expert programmers.
- NiosII SBT for Eclipse. An easy-to-use GUI that offers project management tools, automates build and makefile management, integrates a text editor and compiler, debugger, the NiosII flash programmer, and the Quartus II Programmer. Software application templates included in the GUI make it easy for new software programmers to get started quickly.

As the console-based environment will not be used in this project, the NiosII SBT for Eclipse and NiosII EDS terms will be used without distinction.

The NiosII EDS allows creating and managing single-threaded programs as well as complex applications based on RTOS and/or middleware libraries, available from Altera and third-party vendors.

The NiosII EDS builder is a powerful tool, but a complete explanation of their capabilities is outside of the scope of this project, as there is a large amount of documentation available, so the short explanation will be focused on how to correctly plan the software design using this tool, and how NiosII EDS allows to create NiosII applications quickly and independently of the low-level hardware details.

B.4.2 Embedded software architecture for NiosII Processor. Software layers introduction

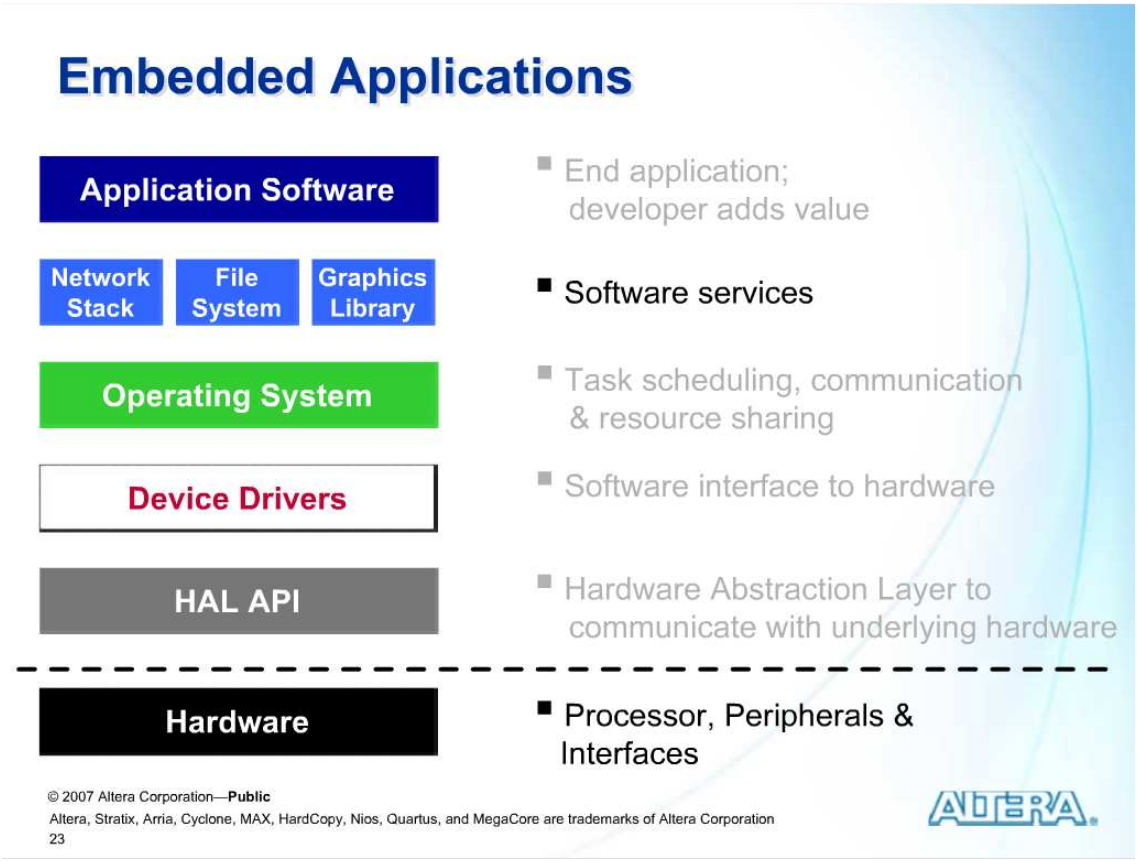


Figure Annex B.8. Embedded applications different abstraction layers

Direct interaction:
HW ⟷ Application SW

In the most basic embedded projects there is an application directly interacting with the hardware platform. This structure maybe is the optimal in terms of hardware usage but lacks a lot of flexibility, as any minor hardware change, like different settings in the NiosII processor or any other added feature, will lead to a complete redevelopment of software. As a consequence, another major issue is that software developer needs a low-level knowledge of the hardware platform to develop the application software and in complex systems that can combine any number of SOPC Builder components, it can lead to a major scalability problem.

As one of main advantages of an FPGA design is that hardware can be changed frequently, it is not recommended except in very special applications to attack directly to hardware from application software.

Adding Hardware Abstraction Layer:
HW ⟷ HAL + Device Drivers ⟷ Application SW

A way to avoid this issues is to abstract the hardware using a hardware independent Application Program Interface (API), the Hardware Abstraction Layer (HAL). The HAL is a lightweight UNIX-style API for interfacing to peripherals and provides system services like interrupt handling, timers and device access. The HAL application program interface (API) is integrated with the ANSI C standard library allowing to access devices and files using familiar UNIX-style C library functions, such as printf(), fopen(), fwrite(), etc.

Anyway, the HAL is not enough to separate the application and hardware-specific code, and the next needed thing is the development of some device drivers, a piece of software written to interface a specific system component to the HAL. This approach abstracts hardware into a set of services that operate through a fixed API, and the application developer will not even notice of hardware changes as long as the same services through the same HAL API are still offered.

HAL device driver abstraction provides a clear distinction between application and device driver software. This driver abstraction promotes reusable application code that is resistant to changes in the underlying hardware. In addition, the HAL standard makes it straightforward to write drivers for new hardware peripherals that are consistent with existing peripheral drivers.

- Application Software interacts with system resources either through the C standard library or HAL API, and does not deal with HW-related details,
- Device Drivers are responsible for making system resources available to the application software, and communicate directly with hardware through low-level hardware access macros.

Tight integration between SOPC Builder and the NiosII EDS automates the construction of a HAL instance for every specific hardware.

- SOPC Builder offers pre-written device drivers for all available components.
- After SOPC Builder generates a hardware system, a custom HAL board support package (BSP) that matches the hardware configuration is generated in the NiosII EDS by using the SOPC Information File.
- Changes in the hardware configuration automatically propagate to the HAL device driver configuration, preventing any changes in the system from creating bugs, so application software can be developed and debugged without concern about whether the software matches the target hardware. For example hardware-related software library calls like printf(), that depend of standard output hardware.
- It is not needed to spend any time changing specific hardware interface code after reconfiguring the system.

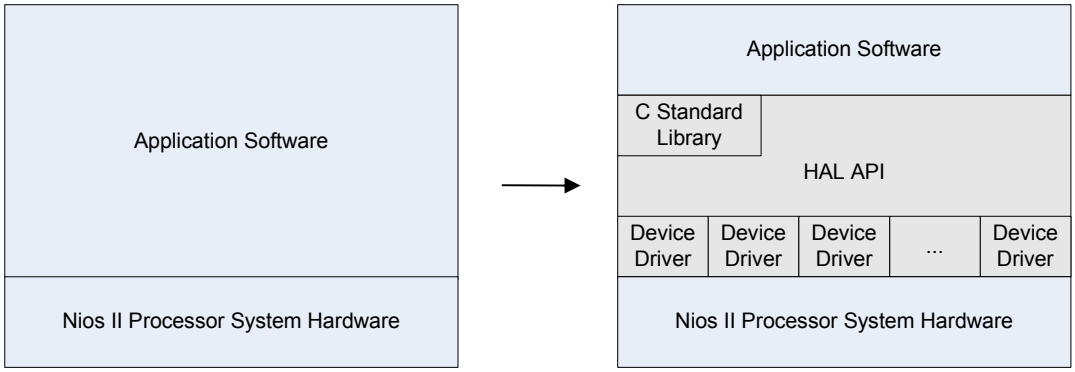


Figure Annex B.9. Adding Hardware Abstraction Layer

Dealing with software complexity by adding an RTOS:
HW ⟷ HAL + Device Drivers + RTOS ⟷ Application SW

Small systems of low complexity are generally called *foreground/background* systems or *super-loops*. It means that the application SW consists of an infinite loop that calls modules to perform the desired operations (background). Interrupt service routines (ISR) handle asynchronous events and perform the critical operations (foreground). The main problem of a super-loop system is that typical execution time is not constant, so the timing of the loop is non-deterministic, and any code change can affect it.
In a complex system where there are several system resources and the desire is to run it in parallel without interfering each other, the need of a Real-Time Operating System (RTOS), also called Real-Time Kernel, arises in order to avoid excessive software complexity. It also allows a better use of resources and the system can be expanded easily without the known limitations of a super-loop system.
Basically an RTOS provides a nice framework to organize different features of the system through some set of useful functionalities:

- Multitasking – scheduling. The process of switching the central processing unit (CPU) between several tasks, maximizing the use of the CPU and providing a useful framework for modular construction of application software by structuring it as tasks with different priorities.
- Context switching. Is the ability of the RTOS to switch from different tasks, saving the current task context to be able to later resume the task, restoring that context when the other task finishes its execution.
- Intertask communication and shared resources access mechanisms generally called kernel services. For resources used by more than one tasks, an RTOS provides mechanisms to ensure mutual exclusion, for example semaphores, in order to prevent data corruption. An RTOS also offers mechanisms to ensure communication between different tasks, like message queues.

In summary, the integration of an RTOS in a large embedded system, can solve a variety of problems that can highly increase complexity of application software, since it provides multitasking capability and allows the application to be broken down into smaller pieces easily, with only the small affordable penalty of more ROM/RAM needed and an additional CPU overhead plus the cost of the RTOS itself.

There are a lot of current RTOS that support NiosII processor, from commercial ones, with a large list of third-party vendors offering support to the popular open source ones like uCLinux. Anyway, Altera distributes a port of MicroC/OS-II (uC/OS-II) for NiosII processors to

provide immediate access to an easy-to-use RTOS that operates on top of the HAL. NiosII EDS offers tools for the construction and configuration of a board support package with uC/OS-II.

- There is no need to modify source files to enable or disable uC/OS-II features as it can be selected with the NiosII EDS
- Pre-written device drivers only used when uC/OS-II environment is selected, are automatically integrated.

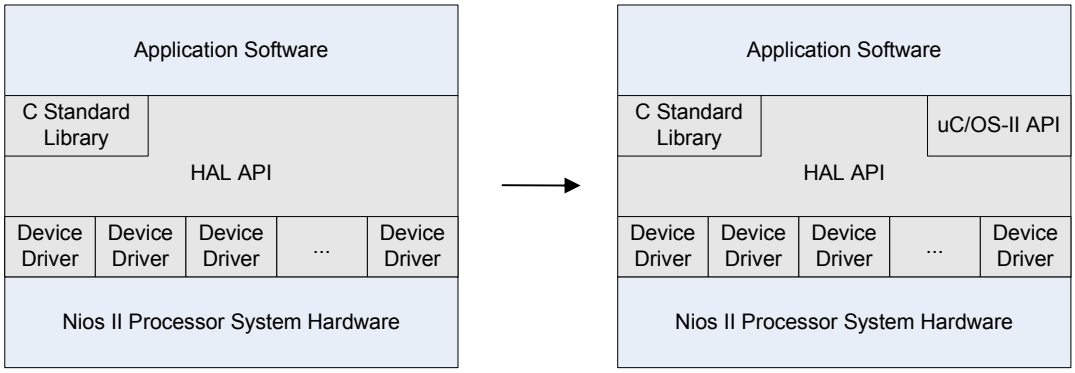


Figure Annex B.10. Adding an RTOS

Adding complex features by using middleware software packages:
HW ⟷ HAL + Device Drivers + RTOS + Middleware ⟷ Application SW

Finally to accelerate the development of the embedded system or to add complex features, it may be decided to use some additional pre-written software services with NiosII support for special functions, for example:

- Network or USB Stacks.
- File Systems.
- Graphics Libraries.

There are several middleware solutions prepared for all the different RTOS that support NiosII processor.
Anyway for easy-to-use access to networking features, Altera distributes two different network stacks integrated with uC/OS-II operating system that also can operate on top of several Ethernet Controllers device drivers, not tying the development to only one different hardware solution

- Lightweight IP (lwIP) TCP/IP stack.
- NicheStack TCP/IP Stack.

These two network stacks are easily configurable from NiosII EDS when creating a custom board support package with uC/OS-II and provide a UNIX-like sockets API to application software to easily add networking capabilities.

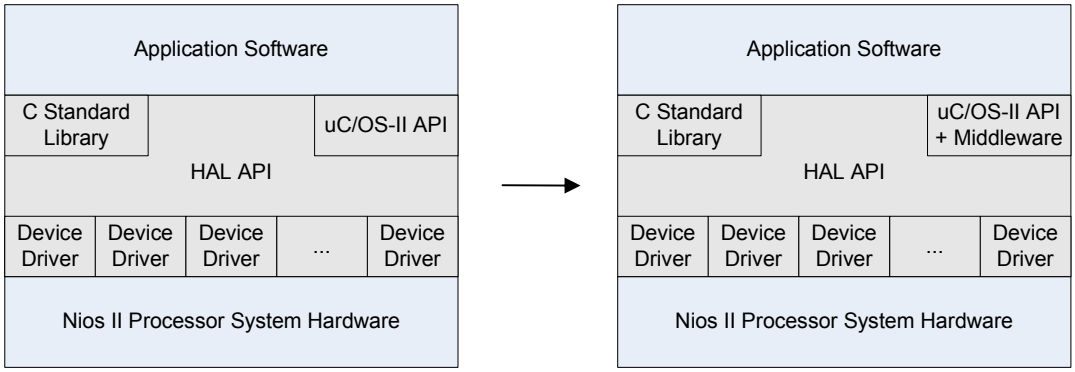


Figure Annex B.11. Adding Middleware to the RTOS

B.4.3 Software Design with NiosII EDS

When the system design phase is completed, the SOPC Information File (.sopcinfo) generated with SOPC Builder will be used to start creating a new NiosII application project. The SOPC Information File is needed to start the software design as it stores the system description, and from this file, NiosII EDS can create a software project designed to target the specific SOPC system. The software design flow can be summarized in four steps

1. Create a customized Board Support Package (BSP) library for the hardware system.
2. Create application software.
3. Build the final project.
4. Download software to target platform and/or start debugging.

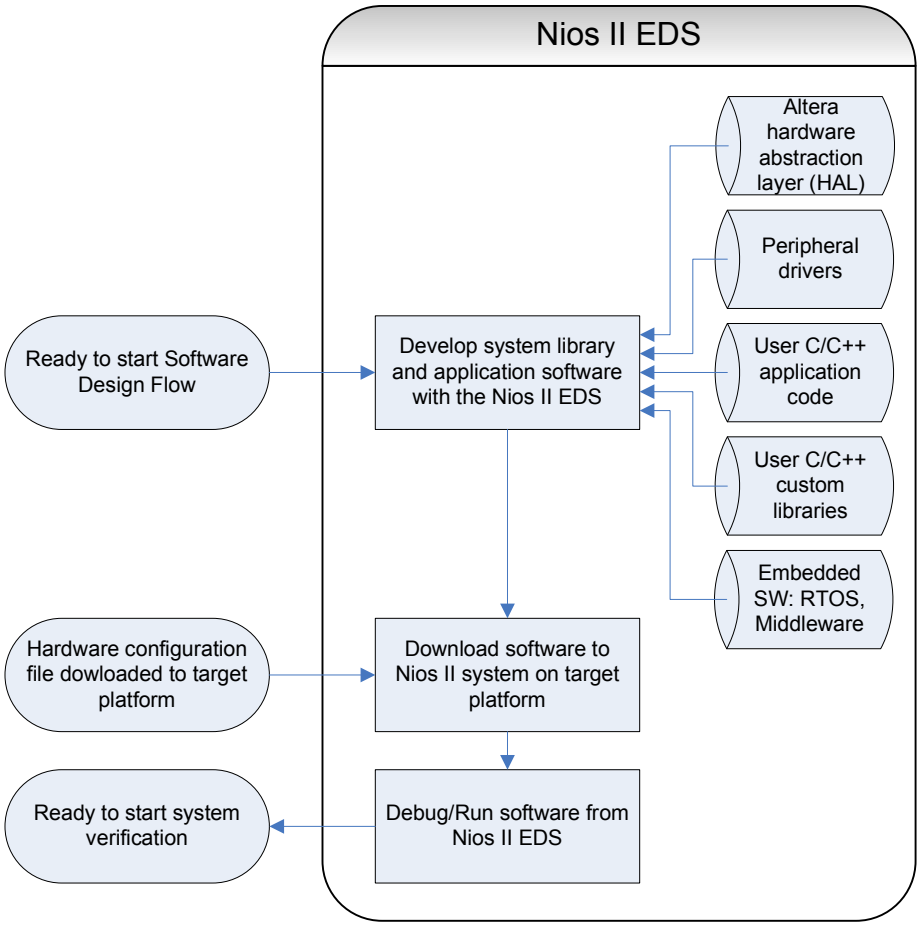


Figure Annex B.12. Software design flow

Creating and configuring the BSP project

The SOPC Information File (.sopcinfo) that contains a system description that the software development tools like the NiosII EDS use to generate a system library, called Board Support Package (BSP), specific to that SOPC Builder system. The NiosII EDS provides tools to modify setting that control the behavior of the BSP. The main elements of the BSP have been explained before when introducing the basics of embedded software, and can be summarized as:

- Hardware Abstraction Layer (HAL) API and Newlib C Standard Library.
- Device Drivers.
- RTOS.
- Middleware optional software packages.

Creating application project

The collection of source code that makes the application project is totally independent of hardware as it only consists of calls to the BSP library and pre-compiled user libraries.

Building final project

The NiosII EDS software provides a project management environment to easily interact between the application and BSP/system library projects that are displayed in different sections of the IDE, as shown in Figure Annex B.13. NiosII EDS also manages the makefiles for the complete build of NiosII software project. BSP makefiles are based on operating system settings, BSP settings, selected software packages, and selected drivers. The application makefile compiles the source code and links it with the BSP and one or more

optional libraries, to create one .elf file that can be downloaded directly to the NiosII processor already configured on the target platform.

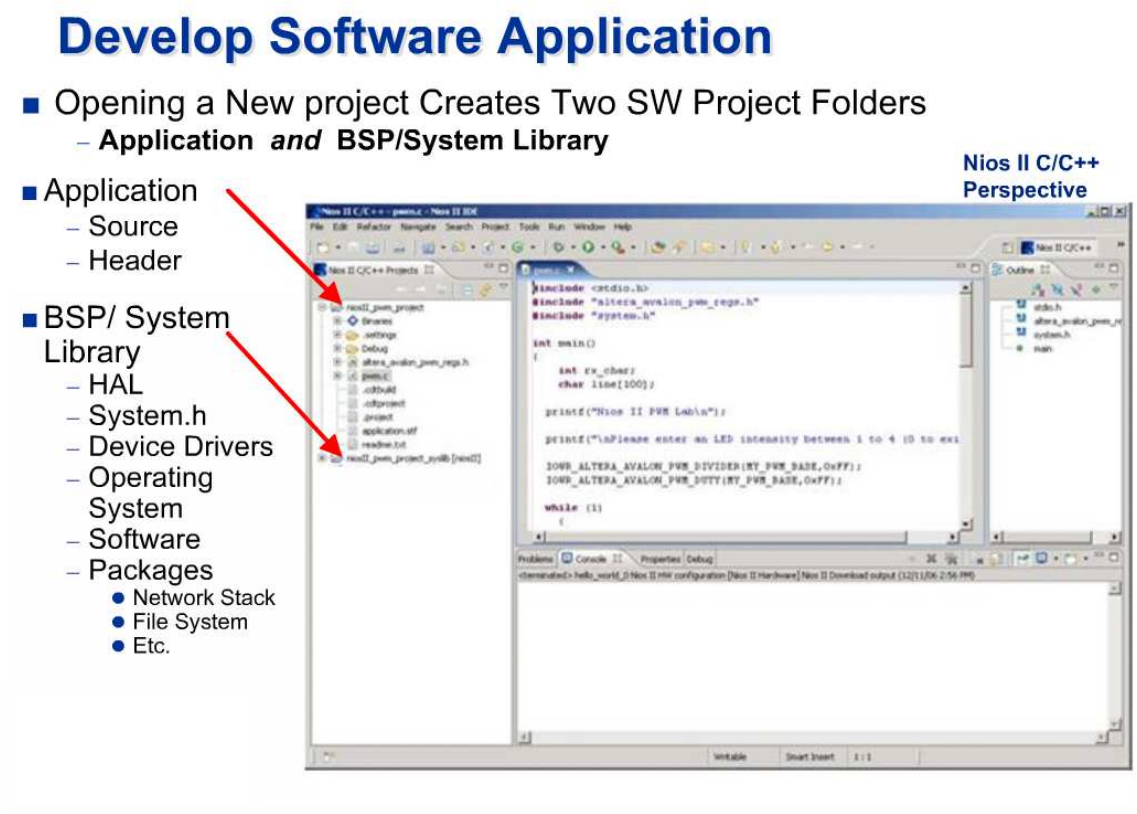


Figure Annex B.13. Application and BSP projects differentiation in NiosII EDS suite

Download & Debug Software

To reach this phase the hardware design flow must be finished to have a fully working target platform where to run the software. The Executable and Linking Format File (.elf) result of compiling the application project can be downloaded directly to the NiosII processor. Besides downloading software to the board, NiosII EDS also provides software debugging options that allows starting and stopping processor execution, set breakpoints and execute code step by step.

B.5 Non-linear Design Flow

Although the design flow diagram is described as a linear process, the development flow has not to be strictly linear, as the system can be gradually designed from a simpler one to the fully completed system.

Also, after running software on target board, maybe some new necessities appear and NiosII system can be redesigned for higher performance, in this case it is possible to return to the hardware design flow to add acceleration logic, for example custom instructions, in order to minimize processor workload in complex software algorithms, off-loading these tasks to hardware, and fully using the possibilities of a FPGA-based system.

Annex C

Proof-of-Concept Prototype main technologies involved

C.1 High-Definition Multimedia Interface (HDMI) Fundamentals

HDMI (High-Definition Multimedia Interface) is the first and most expanded in the industry, uncompressed, all-digital, audio/video interface. Basically it is the digital alternative to typical consumer analog standards, like composite video, component video, VGA or SCART, and provides a digital interface between any audio/video source, such as a set-top box, DVD player, or A/V receiver and an audio and/or video monitor, such as a digital television (DTV), over a single cable.

Architecture and physical link overview
HDMI system architecture is defined by two types of HDMI devices, Sources and Sinks. A given device may have one or more HDMI inputs and one or more HDMI outputs. Each HDMI input on these devices shall follow all of the rules for an HDMI Sink and each HDMI output shall follow all of the rules for an HDMI Source. An HDMI device can be at the same time source and sink if it has both inputs and outputs. These devices are called Repeaters.

As shown in Figure Annex C.1 block diagram, the HDMI cable and connectors carry four differential pairs that make up the Transition Minimized Differential Signaling (TMDS) data and clock channels. These channels are used to carry video, audio and auxiliary data from source to sink devices.

In addition, HDMI carries a Display Data Channel (DDC), a standard I^2C bus communication channel used for configuration and status exchange, between source and sink. All the HDMI sinks are required to have an internal memory data structure, called Enhanced Extended Display Identification Data (E-EDID), which is read by the source to know audio/video capabilities of the sink device. The DDC channel is also used in the High-Bandwidth Digital Content Protection (HDCP) protocol when transmitting data from source to sink.

The CEC protocol bidirectional line provides high-level control functions between a network of audiovisual products connected by HDMI.

The optional utility line, also called HDMI Ethernet and Audio Return Channel (HEAC) line, is one of the new features of the new 1.4 standard providing the possibility of an Ethernet compatible data network between connected devices and also an Audio Return Channel in the opposite direction of the TMDS channels, it means from sink to source, solving the need of external high quality audio cables from sinks to sources.

The HPD line is primary used by the source to detect a working sink is ready, and the DDC communication can start. HDMI 1.4 specification *[20]*, also defines how it can be used for the HDMI Ethernet channel in conjunction with the utility line.

Finally although it is not shown in the block diagram there is an extra +5V signal from the source to the sink typically used by the sinks to detect the connection of a valid source. A typical implementation of HPD line is to tie with a serial resistor the +5V signal from source to the HPD line.

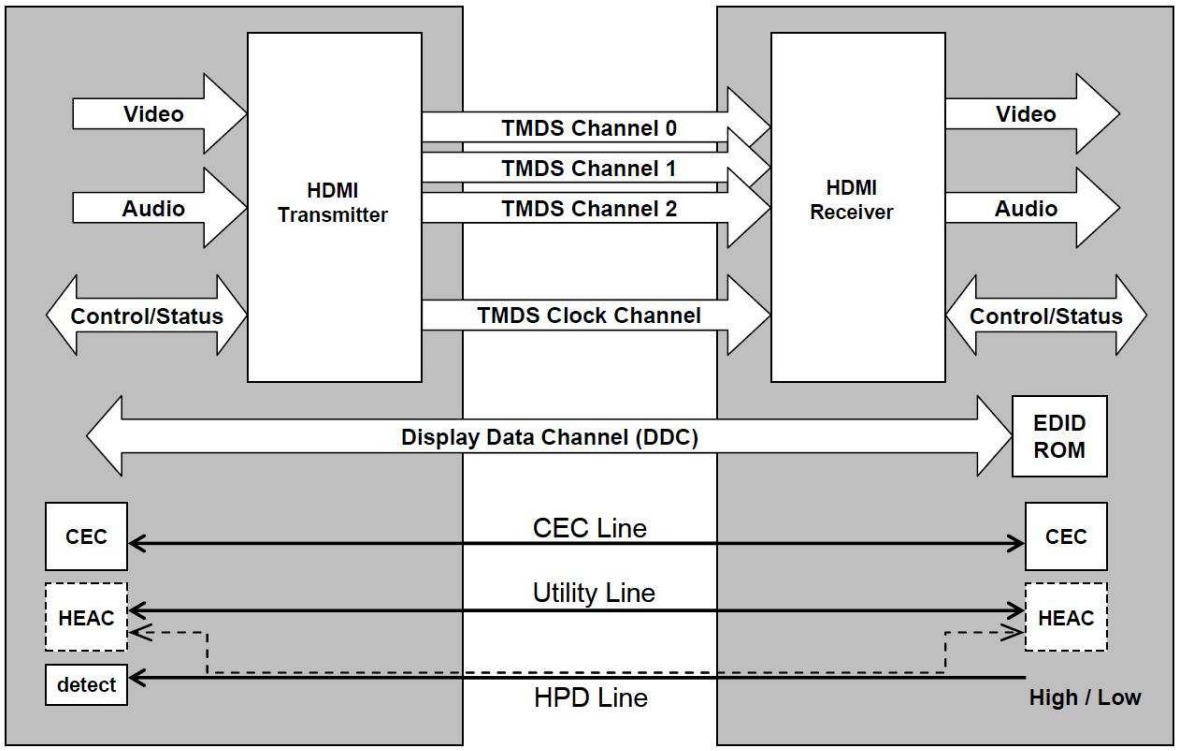


Figure Annex C.1. HDMI block diagram view of communication channels between source and sink devices

Link architecture and operating modes

As explained in the introduction, an HDMI link includes three TMDS data channels and a single TMDS clock channel. The TMDS clock channel runs at a rate proportional to the pixel rate of the transmitted video. In every cycle of the of the TMDS clock channel, each of the three TMDS encoder/serializer transmit a 10-bit character.

Figure Annex C.2 shows a simplified internal block diagram of the encoder/serializer inside an HDMI transmitter in the source, and the decoder/deserializer inside the HDMI receiver in the sink.

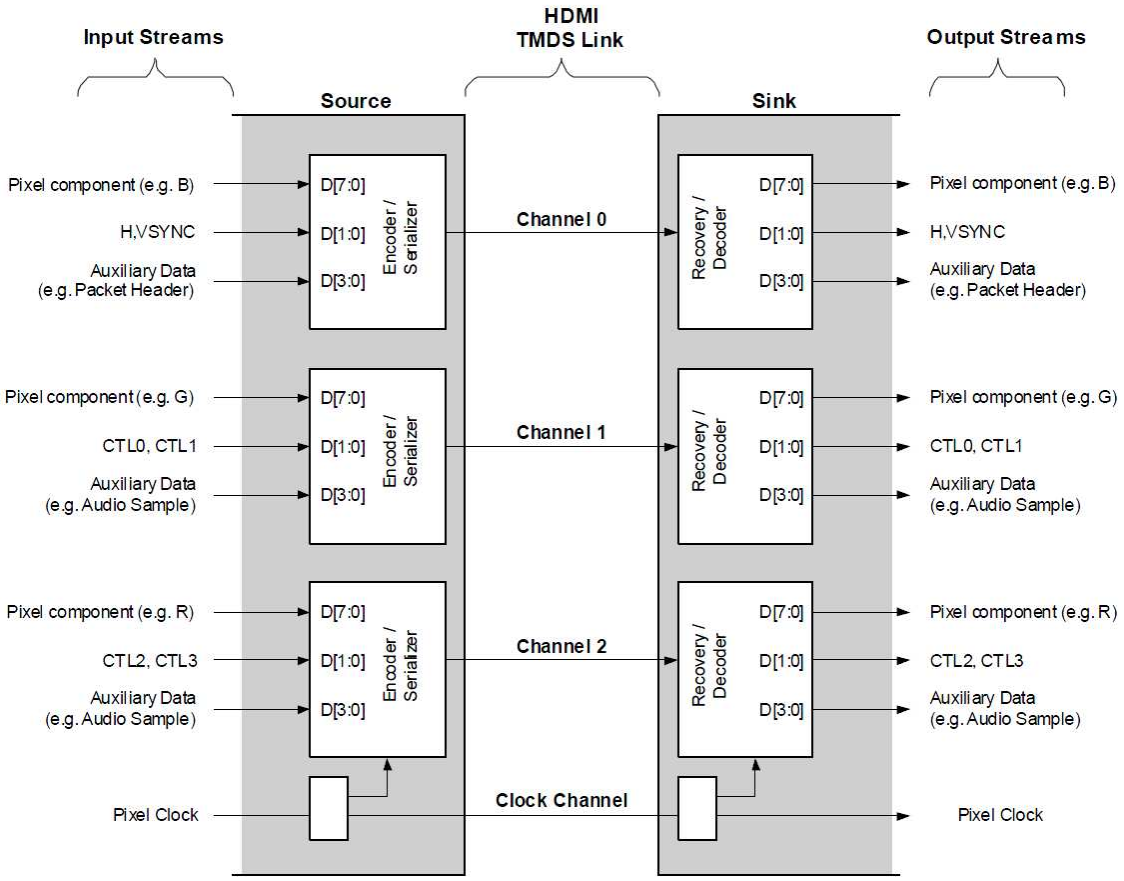


Figure Annex C.2. HDMI encoder/decoder blocks inside HDMI transmitter/receiver devices

The HDMI link operates in one of three modes, Video Data Period, Data Island Period and Control Period, depending on the kind of data the stream is encoding.
During the Video Data Period, the active pixels of an active video line are transmitted. During the Data Island Period, audio and auxiliary data are transmitted using HDMI packets. The Control Period contains the needed signaling information when no video/audio/auxiliary data is transmitted. The Annex Table C.1 shows the encoding type used and data transmitted during each operating mode.

Period	Data Transmitted	Encoding Type
Video Data	Video Pixels	Video Data Coding (8 bits converted to 10 bits)
	(Guard Band)	(Fixed 10 bit pattern)
Data Island	Packet Data - Audio Samples - InfoFrames HSYNC, VSYNC	TERC4 Coding (4 bits converted to 10 bits)
	(Guard Band)	(Fixed 10 bit pattern)
Control	Control - Preamble - HSYNC, VSYNC	Control Period Coding (2 bits converted to 10 bits)

Annex Table C.1. Encoding Type and kind of data transmitted in each operating mode

More details about the HDMI signaling and encoding are not needed to follow the development of this project, anyway more information can be found in the HDMI 1.4 specification *[20]*.

Video
HDMI allows a lot of video formats to be transmitted and displayed, either standard or vendor-specific, but to maximize interoperability between products common DTV formats are defined in the CEA 861-E specification. The video format timings define the pixel and line counts and timings, synchronization pulses position, polarity and duration, and whether the format is interlaced or progressive.
More detailed information about video over HDMI is introduced in the *Chapter 9.2 - Video generation*

Audio
HDMI allows to format audio from different sources and formats in the Audio Sample Packets or in the High Bitrate (HBR) stream packets. These audio packets are transmitted in the TMDS link during the blanking periods of video, in the Data Island Periods.
Main problem is that audio data across the link, which is transmitted at the TMDS clock rate corresponding to the pixel clock rate, does not retain the original audio sampling clock, so the audio clock has to be regenerated. The HDMI clock regeneration architecture is based on finding the relationship expressed by an integer fraction between video and audio clocks. The clock regeneration architecture can be appreciated in Figure Annex C.3. The N and CTS values that express the relationship between the two clocks are transmitted from source to the sink using the special Audio Clock Regeneration packet.

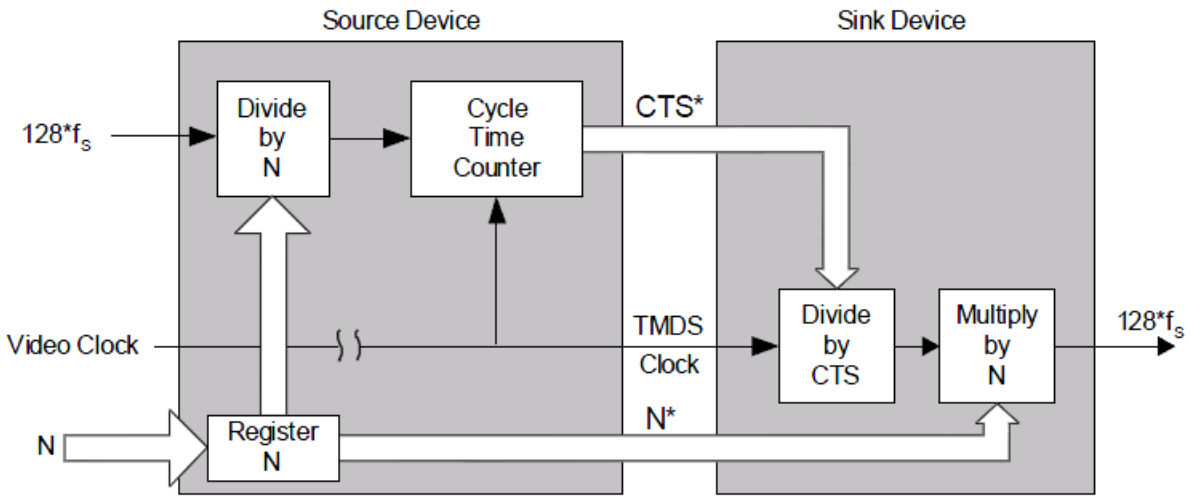


Figure Annex C.3. Audio Clock Regeneration model

More detailed information about audio carried across the HDMI link is introduced in the *Chapter 9.4 - I2S generation*

Control and configuration

The DDC channels is used by an HDMI source to determine the capabilities and characteristics of the sink, by reading the EDID data structure, and the HDMI sources are expected to only deliver the audio and video formats supported by the sink. In addition the HDMI sinks are expected to detect InfoFrames and other packets containing auxiliary data sent by the sink in order to process the audio and video appropriately.
Seven of the sixteen packet types described in the HDMI 1.4 specification deal with audio data, while the other nine packet types deal with auxiliary data.

Content protection (HDCP)

Content protection capability is recommended for all HDMI compliant devices.
HDCP was designed to protect the high-definition video signals as well as HD audio signals.
As a Digital Rights Management (DRM) strategy, HDCP provides three main functions:

- Authentication of devices that are allowed to play HD content.
- Encryption and transmission of HD content between authenticated devices.
- Ability to revoke a device's permission to authenticate.

More detailed information about HDCP protocol can be found in the provided references *[31][32][33][34]*.

C.2 Digital Video concepts

CEA-861-E specification

In order to maximize interoperability between products, a common framework to send digital video needs to be defined, in order to avoid situations of incompatibility between video formats send by the video source and the accepted ones by the video sink. This is the main role of the CEA-861-E specification *[19]* as it establishes protocols, requirements and recommendations for the utilization of uncompressed digital interfaces by consumer electronic devices. It is important to remark that this specification is independent from the digital physical interface, such as DVI or HDMI used in TV or set-top boxes or Open LVDS Display Interface (LDI) typically used in the interface between video main boards and LCD panels. All that requirements for video sources and sinks are finally summarized in the video formats, which are the main output of the specification.

A video format is defined as a group of information required by the sink to properly display an image. Usually it contains some mandatory information and other optional:

- Video Timing. (Mandatory) It can be defined as the waveform associated with the video format. Video timings are further introduced later in this chapter.
- Picture Aspect Ratio. (Mandatory) Ratio of width to height dimension of the picture. CEA-861-E only accepts 16:9 and 4:3
- Color Space (Optional). There are many ways of representing color, and the information about mapping and color model must be known for any video format received by the sink. Typical color spaces are RGB with three color component samples, or YCbCr with luma component and the chroma components. Color space theory on its own is a vast field to explore, and can be further expanded in the provided references *[16], [17]*.
- Quantization Range (Optional). Black and white extreme levels for color components shall be either “Full Range”, including all code values, or “Limited Range”, excluding some values at the extremes. The “Limited Range” exists to eliminate “wrong colors” due to overshoot and undershoot in an analog video signal source, that can be produced by noise, gain variations or transients produced by filtering. The coding ranges issue can be further explored in the reference *[17]*.
- Component Depth (Optional). The number of bits used to represent a color component sample, RGB or YCbCr.
- Subsampling. Is the practice of encoding pictures by implementing less resolution of some color component in front of the others. The subsampling scheme is usually expressed as a three-part ratio, with the color mapping before. For example RGB 4:4:4 really means no subsampling as each one of the three components use the same sample rate. Subsampling is usually used withe the YCbCr color space, and for example chroma subsampling YcbCr 4:2:2 is used in digital video systems because at normal viewing distances, human eye is more sensitive to brightness than color differences, and the bandwidth save incurred when subsampling the chroma by a factor of two, is worth the loss of quality.

Digital Video Concepts

As its root, a video signal is basically just a two-dimensional array of intensity and color data, a still image that is updated at a regular frame rate, conveying the perception of continuous motion. Special timing information, called vertical sync, is used to indicate when a new image is starting, and each still image or video frame is also composed of scan lines, lines of data that occur sequentially one after another from left to right and down the display in every video

frame. Special timing information, called horizontal sync, is used to indicate when a new line is starting.

Compatibility has always been the key to understand the evolution of video systems, and another important concept to understand is the blanking interval. In the past on conventional cathode-ray tube (CRT) TVs and electron beam modulated by a video signal was actively "painting" the image on the screen in a top-to-bottom, left-to-right fashion, by illuminating the phosphors on the screen. Although synchronization signals defined the line and frame changes, the unavoidable interval needed by the beam to retrace from the end of a line to the next one was called horizontal blanking, and the same to retrace from the end of the frame in the right-bottom corner to the start in the left-top corner.

These synchronization concepts apply to both analog and digital video. Digitizing video consists on both sampling and quantizing an analog video signal, it means that the 2D video frames is divided into small regions, like a grid or raster, and discrete amplitude values are assigned based on the intensities of color space components in each region. These regions called pixels are the smallest picture element addressable in the image.
It can be noted that analog video is already sampled spatially in vertical (discrete number of lines), and temporally (discrete number of frames per second), and the pixel sampling process in digital video (discrete number of pixels) corresponds to a spatial sampling in horizontal along scan lines.

The blanking concept has also been carried away to the digital video although there is no need to reposition the electron beam in newer displays, and now the blanking periods with no active video are used to encode packets of other digital information. It has to be noted that horizontal blanking period is counted in pixels per line, and the vertical blanking period is counted in lines per frame.

Finally another important concept to understand is the progressive vs. interlaced video issue. As video is a series of still images, the normal way to transmit it is one frame after another, and the display will completely update the visual information for every frame, that technique is called progressive scanning. But in early days of television, a technique called interlacing was invented to reduce the amount of information sent for every image. Basically it consists in first transmit a frame with only the odd-numbered lines followed by another one with the even-numbered lines and so on. The vertical resolution is halved in interlaced video compared with progressive video, and it can create artifacts and line flicker, but although the modern displays are progressive, there are still a lot of interlaced video formats very typical between broadcasters.

Although this introduction to digital video covers all the important issues to understand the development of the video generation core, more information can be found in the provided references *[17][18][19]*.

Video Timings Definition in CEA-861-E
After getting familiarity with basic video concepts it will come easier to understand the way CEA-861-E specification is defining each of the available video timings and organizing it in a list of Video Identification Codes (VICs) that a source/sink must be able to generate in order to be compliant with the specification.

It basically consists in specifying the behavior of the three involved signals, HSYNC for the start of a new line, VSYNC for the start of a new frame, and DEN to indicate the active video period.

With little exceptions, the set of waveforms is generally divided in 4 different groups:

1. General progressive video format timing (negative sync)
2. General progressive video format timing (positive sync)
3. General interlaced video format timing (negative sync)
4. General interlaced video format timing (positive sync)

Each of the timings is specified with an associated waveform, the number of blanking pixels, active pixels, blanking lines and active lines. The number of blanking pixels/lines is also separated in three zones, the front porch, the period between the last line/frame and the sync pulse leading edges, the sync pulse that can be asserted during several pixels/lines, and the back porch, the period between the end of the sync pulse and the start of active video.

Figure Annex C.4 shows how each of the timings is specified with the pixel frequency, the number of blanking pixels, active pixels, blanking lines and active lines. This table also specifies for every VIC the type of scanning, progressive or interlaced. The horizontal and vertical frequencies are also included as a reference.

Field Rate[5]									(kHz)	(Hz)	(MHz)
	VIC	Hactive	Vactive	I / P	Htotal	Hblank[5]	Vtotal	Vblank[5]	H Freq[5]	V Freq[4]	Pixel Freq[5]
50Hz	17,18	720	576	Prog	864	144	625	49	31.250	50.000	27.000
	19	1280	720	Prog	1980	700	750	30	37.500	50.000	74.250
	20	1920	1080	Int	2640	720	1125	22.5[1]	28.125	50.000	74.250
	21,22	1440[2]	576	Int	1728[2]	288	625	24.5[1]	15.625	50.000	27.000
	23,24	1440[2]	288	Prog	1728[2]	288	312	24	15.625	50.080	27.000
	23,24	1440[2]	288	Prog	1728[2]	288	313	25	15.625	49.920	27.000
	23,24	1440[2]	288	Prog	1728[2]	288	314	26	15.625	49.761	27.000
	25,26	2880[2]	576	Int	3456[2]	576	625	24.5[1]	15.625	50.000	54.000
	27,28	2880[2]	288	Prog	3456[2]	576	312	24	15.625	50.080	54.000
	27,28	2880[2]	288	Prog	3456[2]	576	313	25	15.625	49.920	54.000
	27,28	2880[2]	288	Prog	3456[2]	576	314	26	15.625	49.761	54.000
	29,30	1440[2]	576	Prog	1728[2]	288	625	49	31.250	50.000	54.000
	31	1920	1080	Prog	2640	720	1125	45	56.250	50.000	148.500
	37,38	2880[2]	576	Prog	3456[2]	576	625	49	31.250	50.000	108.000
	39	1920	1080	Int	2304	384	1250	85	31.250	50.000	72.000
60Hz[3]	1	640	480	Prog	800	160	525	45	31.469	59.940[3]	25.175
	2,3	720	480	Prog	858	138	525	45	31.469	59.940[3]	27.000
	4	1280	720	Prog	1650	370	750	30	45.000	60.000[3]	74.250
	5	1920	1080	Int	2200	280	1125	22.5[1]	33.750	60.000[3]	74.250
	6,7	1440[2]	480	Int	1716[2]	276	525	22.5[1]	15.734	59.940[3]	27.000
	8,9	1440[2]	240	Prog	1716[2]	276	262	22	15.734	60.054[3]	27.000
	8,9	1440[2]	240	Prog	1716[2]	276	263	23	15.734	59.826[3]	27.000
	10,11	2880[2]	480	Int	3432[2]	552	525	22.5[1]	15.734	59.940[3]	54.000

Figure Annex C.4. CEA-861-E Table 2 Video Format Timings – Detailed Timing Information

In the next CEA-861-E table shown in Figure Annex C.5, the number of blanking pixels/lines is separated in three different zones, the front porch, the period between the last line/frame and the sync pulse leading edges, the sync pulse that can be asserted during several

pixels/lines, and the back porch, the period between the end of the sync pulse and the start of active video. This table importantly remark which kind of waveform must be used to generate the final timings in the second column (Fig), specifying also the synchronization pulses polarity for each VIC.

Field Rate	VIC	Fig	Hfront	Hsync	Hback	Hpol[18]	Vfront	Vsync	Vback	Vpol[18]	Ln	Reference Standard	Notes
Low	60	2	1760	40	220	P	5	5	20	P	1	SMPTE 296M [40]	1,2, 25
	61	2	2420	40	220	P	5	5	20	P	1	SMPTE 296M [40]	1,2
	62	2	1760	40	220	P	5	5	20	P	1	SMPTE 296M [40]	1,2
	32	2	638	44	148	P	4	5	36	P	1	SMPTE 274M [2]	14
	33	2	528	44	148	P	4	5	36	P	1	SMPTE 274M [2]	14
	34	2	88	44	148	P	4	5	36	P	1	SMPTE 274M [2]	14
50Hz	17,18	1	12	64	68	N	5	5	39	N	1	ITU-R BT.1358 [56]	
	19	2	440	40	220	P	5	5	20	P	1	SMPTE 296M [40]	1,2
	20	4	528	44	148	P	2	5	15	P	1	SMPTE 274M [2]	1,2
	21,22	3	24	126	138	N	2	3	19	N	1	ITU-R BT.656–4 [54]	6, 15
	23,24	1	24	126	138	N	2^{22}	3	19	N	1	ITU-R BT.1358 [56]	7, 14, 19
	23,24	1	24	126	138	N	3^{23}	3	19	N	1	ITU-R BT.1358 [56]	7, 14, 15, 19
	23,24	1	24	126	138	N	4^{24}	3	19	N	1	ITU-R BT.1358 [56]	7, 14, 15, 19
	25,26	3	48	252	276	N	2	3	19	N	1	ITU-R BT.656–4 [54][17]	8, 13, 14, 15
	27,28	1	48	252	276	N	2^{22}	3	19	N	1	ITU-R BT.656–4 [54][17]	7, 8, 12, 13,19
	27,28	1	48	252	276	N	3^{23}	3	19	N	1	ITU-R BT.656–4 [54][17]	7, 8, 12, 13,19
	27,28	1	48	252	276	N	4^{24}	3	19	N	1	ITU-R BT.656–4 [54][17]	7, 8, 12, 13,19
	29,30	1	24	128	136	N	5	5	39	N	1	ITU-R BT.1358 [56]	9, 10, 14
	31	2	528	44	148	P	4	5	36	P	1	SMPTE 274M [2]	14
	37,38	1	48	256	272	N	5	5	39	N	1	ITU-R BT.1358 [56]	9, 11
	39	5	32	168	184	P	23	5	57	N	1	AS 4933.1-2005 [67]	5
60Hz	1	1	16	96	48	N	10	2	33	N	1	VESA DMT [65]	3, 4
	2,3	1	16	62	60	N	9	6	30	N	7	CEA-770.2-D [31]	2
	4	2	110	40	220	P	5	5	20	P	1	CEA-770.3-C [32]	1,2
	5	4	88	44	148	P	2	5	15	P	1	CEA-770.3-C [32]	1,2
	6,7	3	38	124	114	N	4	3	15	N	4	CEA-770.2-D [31]	2, 15
	8,9	1	38	124	114	N	4^{20}	3	15	N	4	CEA-770.2-D [31][17]	7, 14, 15, 19
	8,9	1	38	124	114	N	5^{21}	3	15	N	4	CEA-770.2-D [31][17]	7, 14, 15, 19
	10,11	3	76	248	228	N	4	3	15	N	4	CEA-770.2-D [31][17]	8, 13

Figure Annex C.5. CEA-861-E Table 3: Video Format Timings – Detailed Sync Information

Only some of the available timings are reproduced in the figures, and the waveform associated with a generic progressive video timing is shown in the Figure Annex C.6. There are another four generic waveforms depending on syncs polarity and line scan of the video timing. This waveform is used by the timings with Fig=1, so, for example this waveform is used for VIC #17 and #18 with 720 active lines, one of the most typical in the market, and commonly known in the HD video industry as 720p@50Hz.

In the CEA-861-E specification *[19]* there is the rest of information with the full contents of the table and the waveforms for the other timing groups like progressive video with positive sync, and interlaced video for both positive and negative sync.

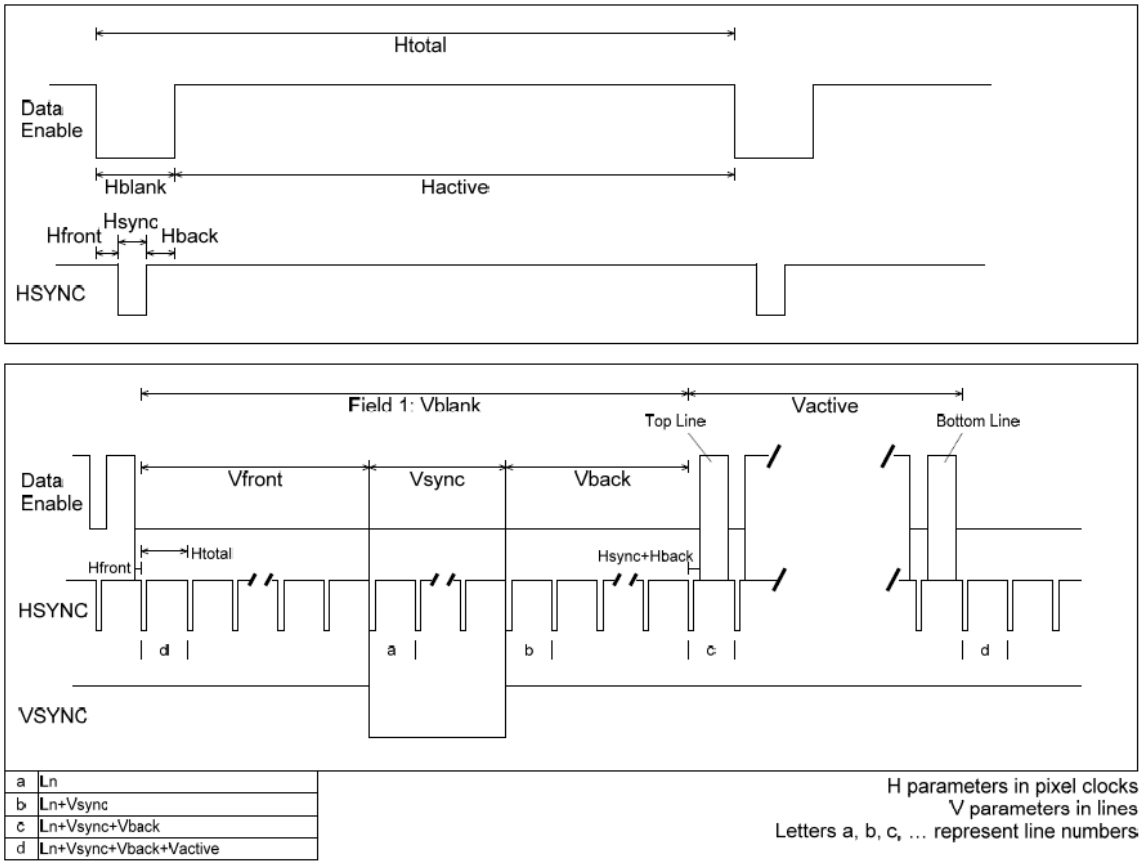


Figure Annex C.6. CEA-861-E Generic waveform for General Video Format Timing (Negative Sync)

C.3 Digital Audio Concepts and I²S

Digital Audio Basics

First of all the basic concepts of digitizing analog audio signals will be introduced.
All A/D and D/A conversions should obey to the Nyquist-Shannon sampling theorem. In short, it says that the analog signals have to be sampled at a rate exceeding twice its highest frequency component in order to be able to reconstruct the original signal in the subsequent D/A conversion. If this condition is not fulfilled aliasing appears in the reconstructed signal. In most of the systems the original signal is band-limited by using a low-pass filter and the sampling rate is twice the higher frequency plus a margin because the non-ideality of these filters. In audio applications, that means the lowest sampling rate that was selected to reproduce music is 44.1KHz, in order to capture the 20Hz-20KHz range of human hearing. The Annex Table C.2 shows some of the common audio sampling rates and its application.

Sampling Frquency	Use
8 kHz	Telephone. Adequate for human speech.
44,1 kHz	Audio CD and commonly used also with MPEG-1 audio.
48 kHz	Professional audio systems.
96 kHz	DVD-Audio, BD-ROM(Blu-ray) audio tracks.
192 kHz	DVD-Audio, BD-ROM(Blu-ray) audio tracks. High-definition audio recording devices and audio editing software.
2,8224 MHz	Super Audio-CD (SACD), 1-bit sigma-delta modulation process known as Direct Stream Digital (DSD).

Annex Table C.2. Typical audio sampling rates and use

The most common method to digitally represent the sampled analog signals is the pulse-code modulation (PCM). A PCM stream is a digital representation of an analog signal, where an analogue signal is sampled at uniform intervals, and each sample is encoded with a digital level. PCM streams have two basic properties that determine their fidelity to the original analog signal: the sampling rate, which is the number of times per second that samples are taken; and the bit depth, which determines the number of possible digital values that each sample can take. The fully discrete representation of the input signal can be easily encoded as digital data for storage and manipulation.

Figure Annex C.7 shows the PCM representation of an analog sine wave converted with an ideal A/D converted with different levels of resolution.

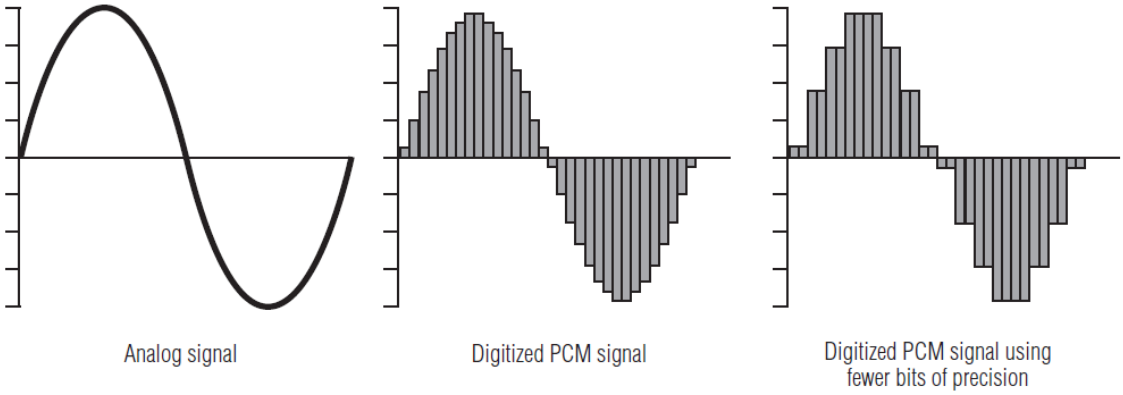


Figure Annex C.7. PCM representation of an analog sine wave

Once the analog data has been digitally encoded, some standard is needed to carry the PCM information between the A/D converter and the audio processing devices. Inter-IC Sound (I^2S) protocol *[40]* is the most common standard used for the digital transmission of audio signals encoded in a PCM stream, using a serial communication bus.

There are other digital audio interconnection standards as Sony/Philips Digital Interconnect Format (S/PDIF) useful to carry PCM audio channels, but for example S/PDIF is oriented to external connections between commercial audio equipment instead of internal buses interconnections.

Finally there are also alternatives to PCM architecture, as in some point the utilization of higher sampling rates and larger bit depths was not enough to continue improving the digitized audio quality. Other digital encoding structures like Direct-Stream Digital (DSD) represent the alternative to PCM, as signal is stored as delta-sigma modulated digital audio, a sequence of single bit values at a frequency sampling rate much higher than the CD Audio sampling rates. For more information about these standards some references are provided *[44]*.

The ADV7511 is capable of receiving audio data in either I^2S, SPDIF, DSD, DST, or HBR format for packetization and transmission over the HDMI interface in the audio packets, but in order to constraint the prototype design, all the effort has been concentrated in the more common audio interface for internal buses, I^2S.

I^2S protocol

The I^2S bus consists of at least three mandatory signals and some optional ones.

- Bit Clock Signal or System Clock Signal (BCK, SCK). (Mandatory). This is the serial audio bit clock and is used by the receiver to store the serial data present on the Data line.
- Word Select Signal or Left/Right Word Clock (WS, LRCK). (Mandatory). In the I^2S bus two channels are transmitted over the same serial bus, using a time-division multiplexing technique. Basically the polarity of this signal indicates the channel being transmitted in the data line. LRCK=0; channel 1 (Left). LRCK=1; channel 2 (Right)
- Data Line (DATA). (Mandatory). I^2S data is the PCM stream transmitted in two's complement with the MSB first. The MSB is transmitted first because the transmitter and receiver may have different word lengths, from 16 to 32 bits per sample. It is not

necessary for the transmitter to know how many bits the receiver can handle and process. The receiver does not need to know the number of bits either

- Master Clock Signal (MCK). (Optional). Some systems with an I²S interface require a master clock signal with a frequency higher than bit clock. For example in some A/D converters is required for operating the digital interpolation filters and multilevel delta-sigma modulators *[41]*
- Extra Data Lines (DATA_1..n). (Optional). There are some multi-channel systems that need more than a two-channel stereo interface, and that can be achieved adding extra data lines each one accommodating two extra audio channels.

Figure Annex C.8 shows the different system configurations depending on which one of the receiver or transmitter acts in master mode, thus generating the clock signals. The most common configuration is when the transmitter acts as the master and the receiver as the slave.

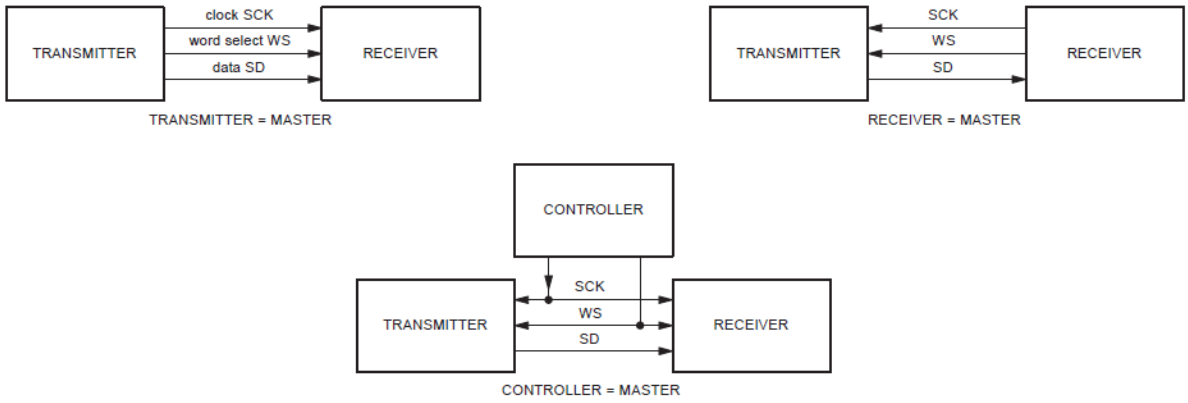


Figure Annex C.8. Different I²S system configurations

I²S clocks relationships, audio data formats and timings

Both LRCK and BCK should be synchronous, and in case a master clock is used, it also has to be synchronous with these two signals.

The different relationships between the clocks can be appreciated at Figure Annex C.9.

- LRCK is operated at the sampling frequency, Fs.
- BCK operates at a frequency which is a multiple of the sampling frequency and depends on the number of bits used to encode every channel and the number of channels, usually 2. So for common precisions of 16,24 and 32-bits per audio sample, the BCK can be operated at 32,48, or 64 times the sampling frequency Fs. The minimum bit rate needed can be calculated as: *Bit rate = (sampling rate) x (bit depth) x (number of channels)*

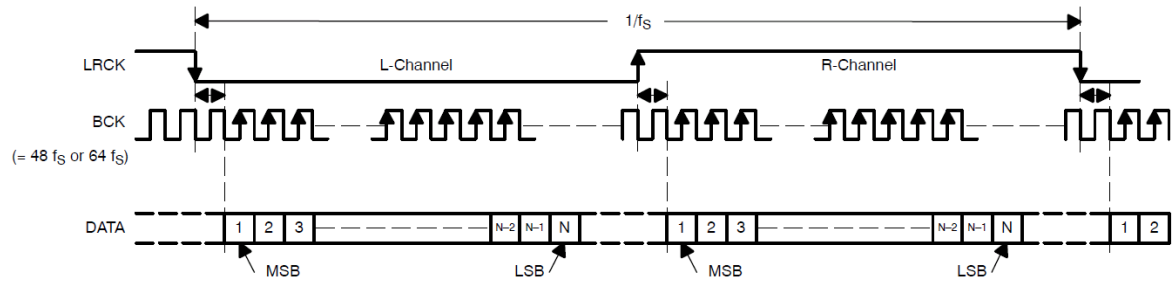


Figure Annex C.9. Standard I²S Data Format

Figure Annex C.10 shows a timing diagram with BCK operating at 64 Fs, and that means a maximum word length of 32 bits. Transmitting MSB first allows both the transmitter and receiver to not care what the audio precision of the remote device is.
If the transmitter is sending 32 bits per channel to a device with only 24 bits of internal precision, the receiver may simply ignore the extra bits of precision by not storing the bits past the 24th bit. Likewise, if the transmitter is sending 16 bits per channel to a receiving device with 24 bits of precision, the receiver will simply zero-fill the missing bits. This feature makes it possible to mix and match components of varying precision without reconfiguration.

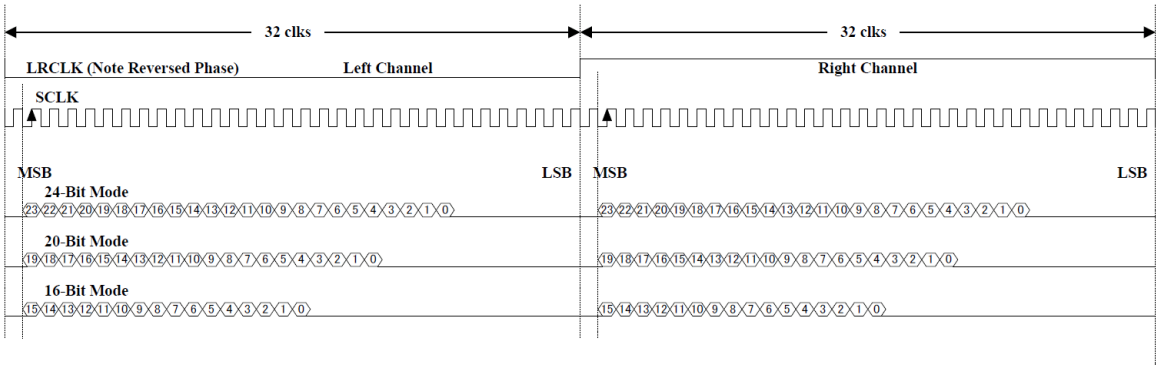


Figure Annex C.10. Standard I^2S Data Format with BCK=64Fs and several audio sample bit depth

In the previous timing diagrams it can be appreciated than in the standard I^2S Data Format, LRCK signal changes one clock period before the MSB is transmitted, and data is valid on the rising edge of bit clock. That is the expected behavior in the industry standard I^2S, but there are other audio data formats with slim differences.

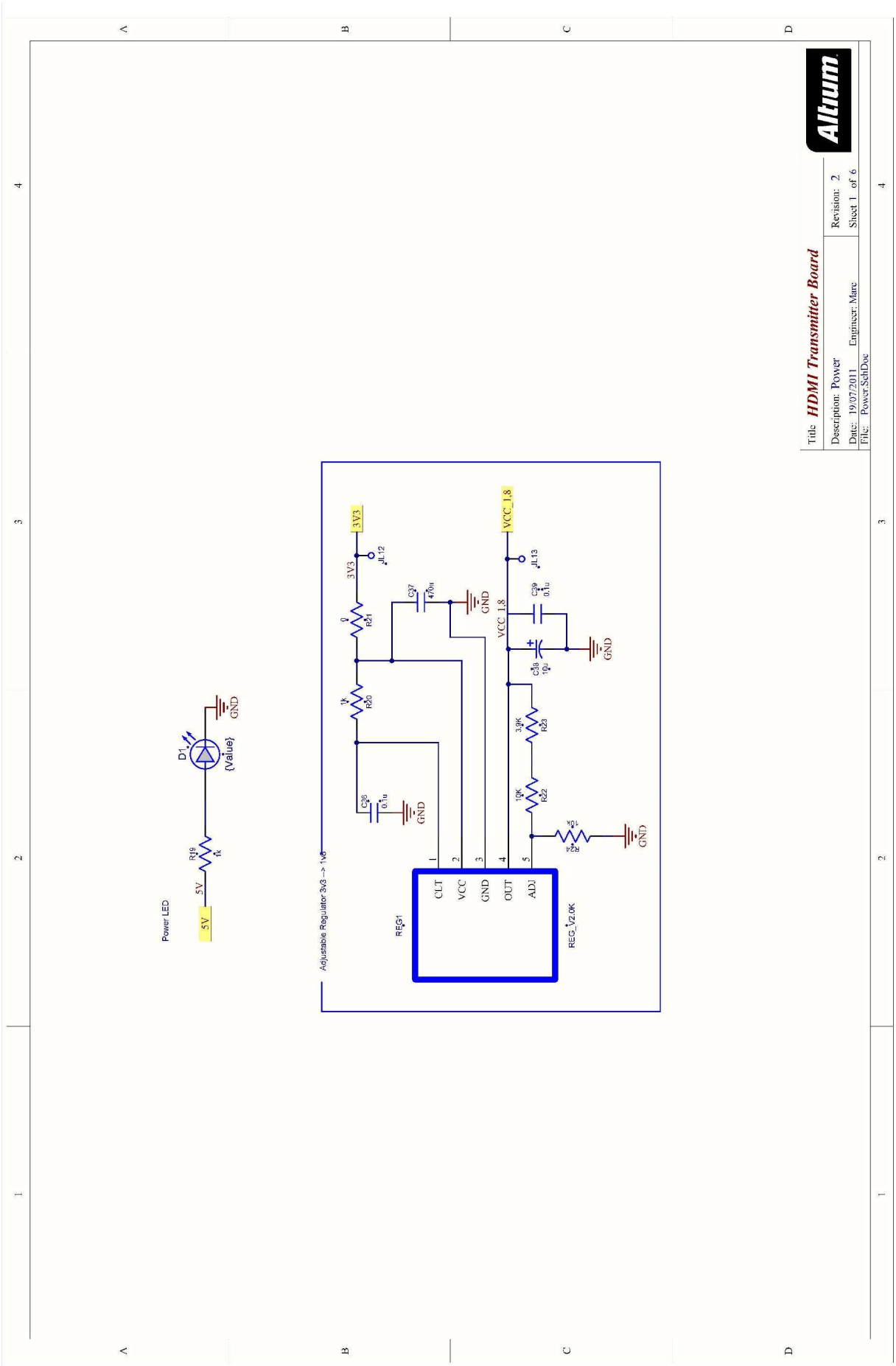
In Left-Justified Data Format, the MSB of data appears on the data lines at the same time the LRCK toggles. If maximum word length is greater than the audio sample precision, the extra trailing bits after the LSB are zero-filled.

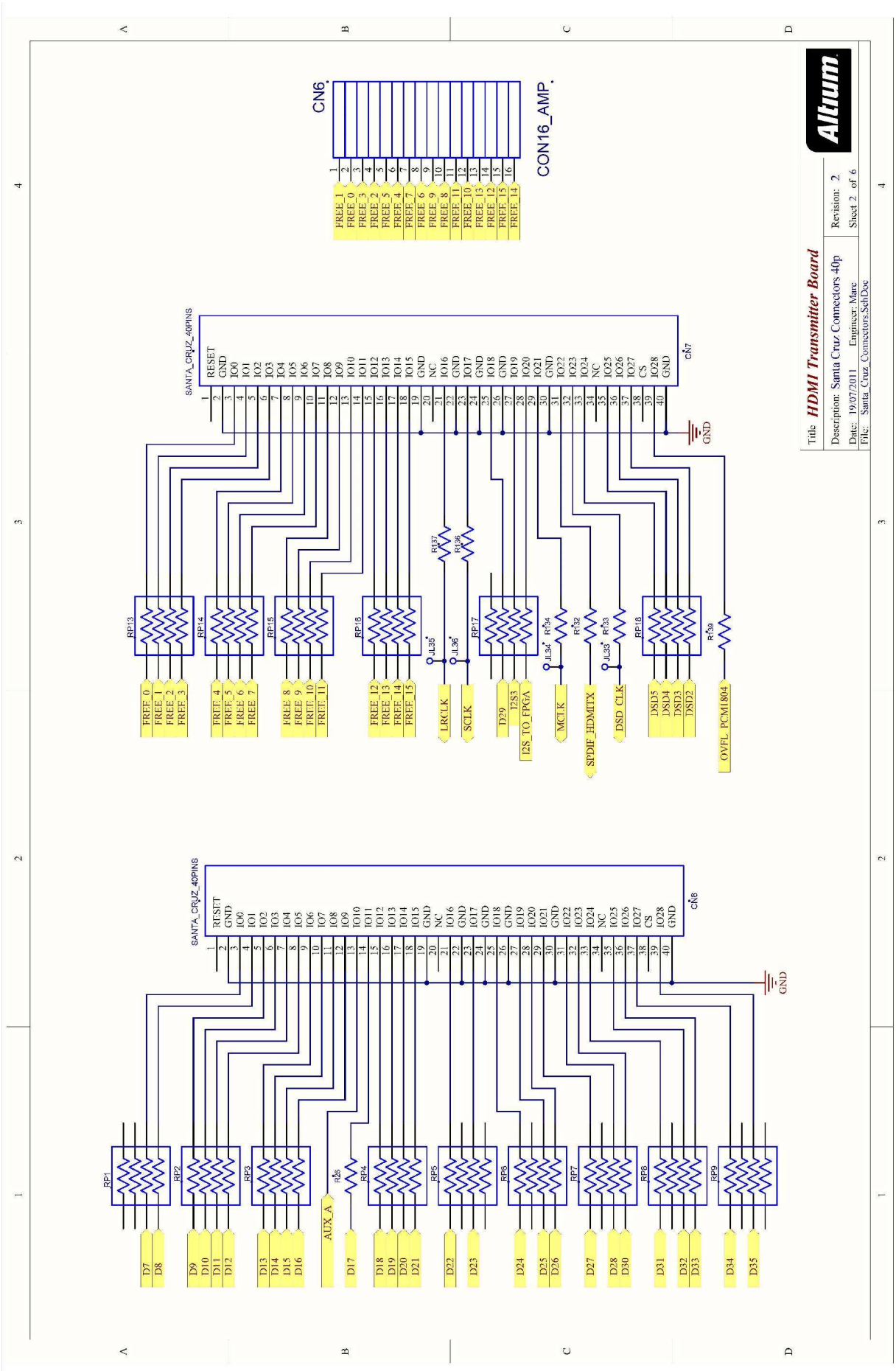
In Right-Justified Data Format, the LSB of data is always clocked by the last bit clock before the LRCK transition. If maximum word length is greater than the audio sample precision, the extra leading bits before the MSB are zero-filled.

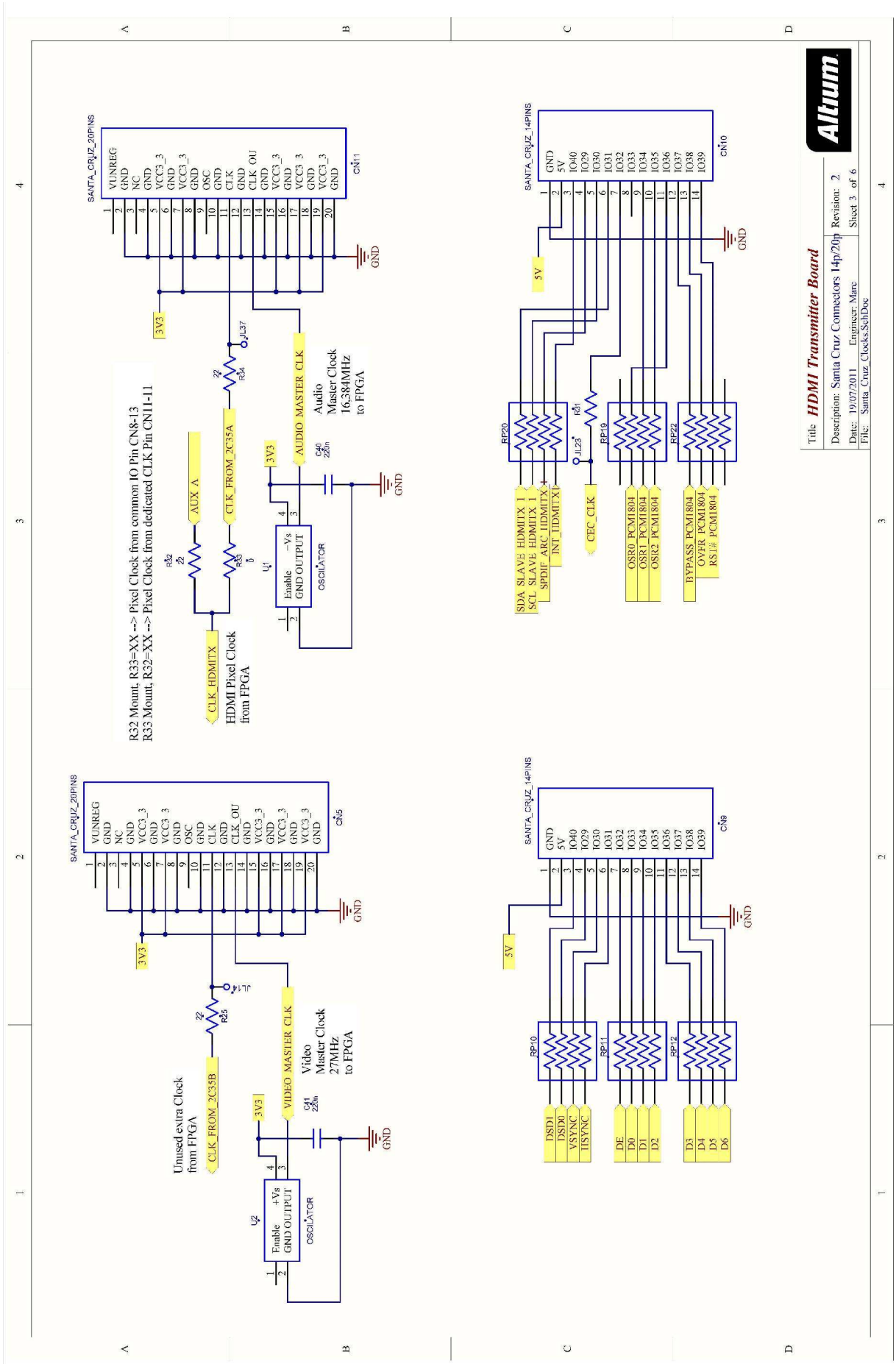
Annex D

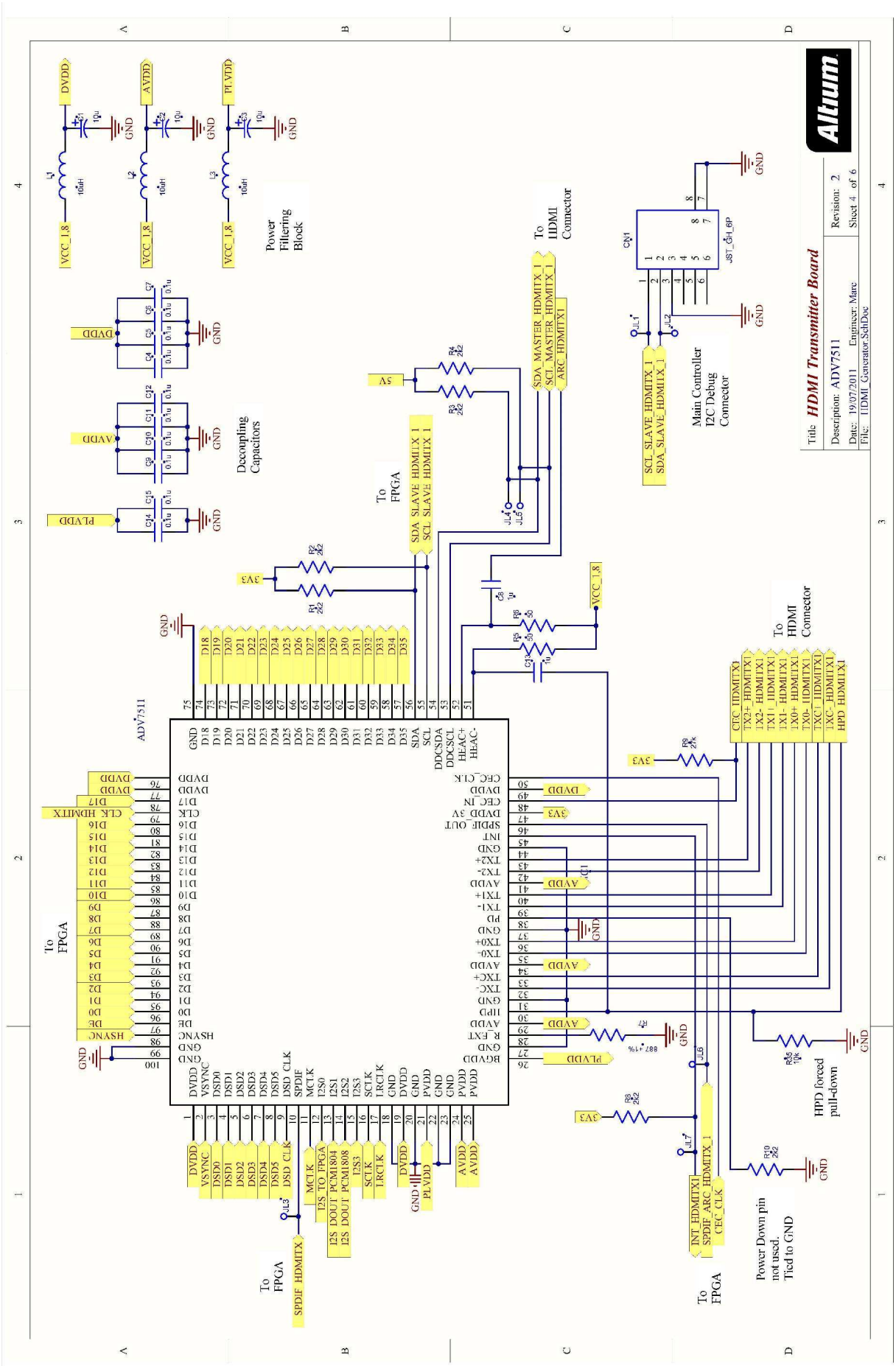
HDMI Transmitter Board Schematics

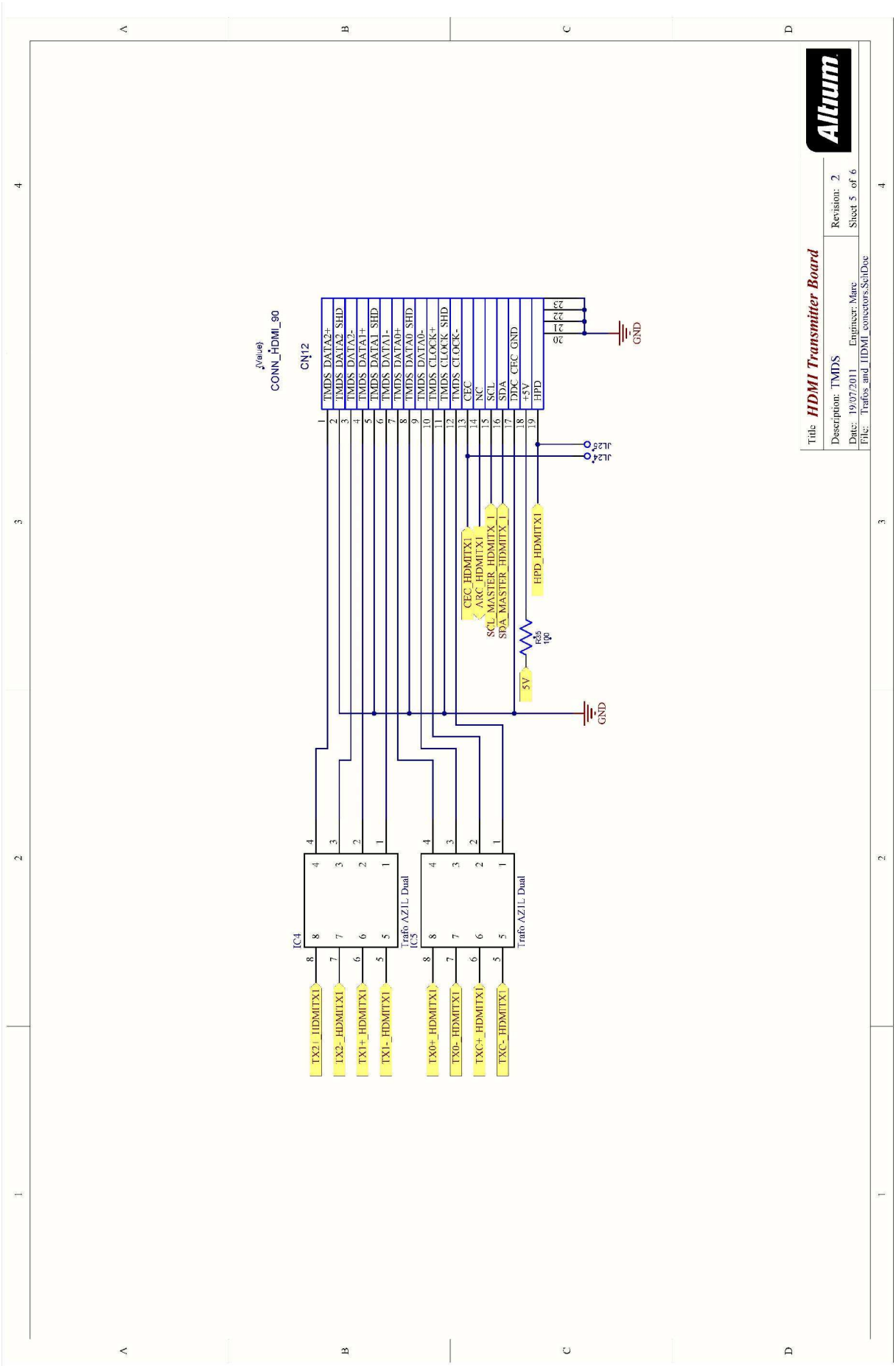
In the next pages the schematics of the HDMI transmitter board are attached to the document.

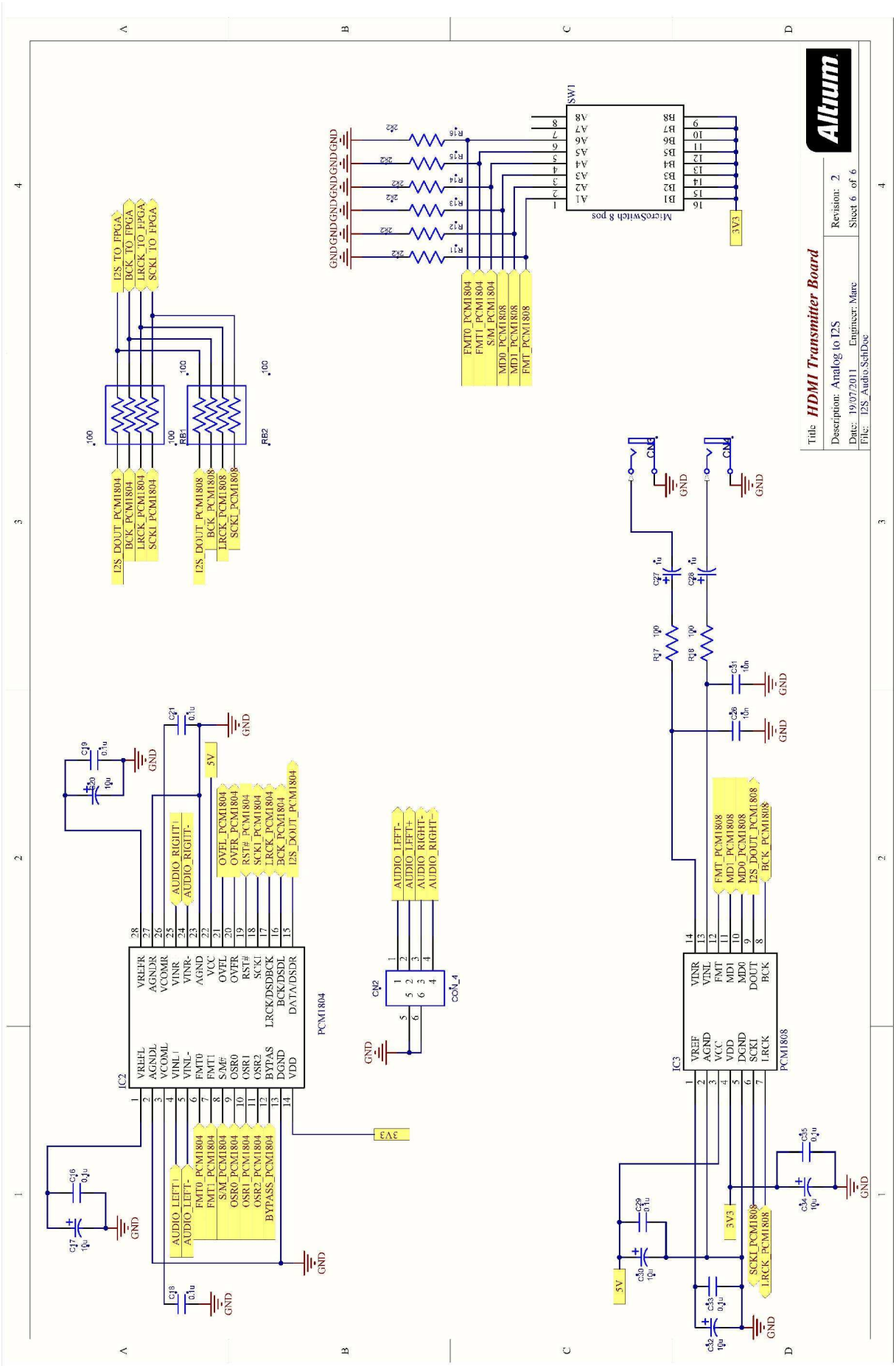

Title HDMI Transmitter Board
Description: Power
Revision: 2
Date: 19/07/2011
Engineer: Marc
Sheet 1 of 6
File: Power.SchDoc
Altium
Power LED
D1
{Value}
R19
1k
5V
GND
Adjustable Regulator 3v3 --> 1v8
REG1
CUT
VCC
GND
OUT
ADJ
REG_V2.0K
3V3
VCC_1.8
R21
0
R20
1k
R23
3.9K
R22
1.9K
C36
0.1u
C37
470u
C38
10u
C39
0.1u


CN6
CON16_AMP
SANTA_CRUZ_40PINS
CN7
CN8
RP1
RP2
RP3
RP4
RP5
RP6
RP7
RP8
RP9
RP13
RP14
RP15
RP16
RP17
RP18
R137
R136
R134
R132
R133
R139
JL35
JL36
JL34
JL33
AUX_A
GND
LRCLK
SCLK
I2S_TO_FPGA
MCLK
SPDIF_HDMITX
DSD_CLK
OVFL_PCM1804
Altium
HDMI Transmitter Board
Description: Santa Cruz Connectors 40p
Revision: 2
Date: 19/07/2011
Engineer: Marc
Sheet 2 of 6
File: Santa Cruz Connectors.SchDoc


SANTA_CRUZ_20PINS
SANTA_CRUZ_14PINS
R32 Mount, R33=XX --> Pixel Clock from common IO Pin CN8-13
R33 Mount, R32=XX --> Pixel Clock from dedicated CLK Pin CN11-11
HDMI Pixel Clock from FPGA
Audio Master Clock 16.384MHz to FPGA
Video Master Clock 27MHz to FPGA
Unused extra Clock from FPGA
OSCILATOR
Title HDMI Transmitter Board
Description: Santa Cruz Connectors 14p/20p
Revision: 2
Date: 19/07/2011
Engineer: Marc
Sheet 3 of 6
File: Santa_Cruz_Clocks.SchDoc
Altium


Power Filtering Block
Decoupling Capacitors
To FPGA
To HDMI Connector
To IIDMI Connector
Main Controller I2C Debug Connector
HPD forced pull-down
Power Down pin not used. Tied to GND
ADV7511
Title HDMI Transmitter Board
Description: ADV7511
Date: 19/07/2011
Engineer: Marc
File: IIDMI_Generator.SchDoc
Revision: 2
Sheet 4 of 6


CONN_HDMI_90
CN12
TMDS_DATA2+
TMDS_DATA2_SHD
TMDS_DATA2-
TMDS_DATA1+
TMDS_DATA1_SHD
TMDS_DATA1-
TMDS_DATA0+
TMDS_DATA0_SHD
TMDS_DATA0-
TMDS_CLOCK+
TMDS_CLOCK_SHD
TMDS_CLOCK-
CEC
NC
SCL
SDA
DDC_CEC_GND
+5V
HPD
GND
CEC_HDMITX1
ARC_HDMITX1
SCL_MASTER_HDMITX_1
SDA_MASTER_HDMITX_1
HPD_HDMITX1
5V
R85
100
IC4
IC5
Trafo AZ1L Dual
TX2+_HDMITX1
TX2-_HDMITX1
TX1+_HDMITX1
TX1-_HDMITX1
TX0+_HDMITX1
TX0-_HDMITX1
TXC+_HDMITX1
TXC-_HDMITX1
Altium
Title HDMI Transmitter Board
Description: TMDS
Revision: 2
Date: 19/07/2011
Engineer: Marc
Sheet 5 of 6
File: Trafos and HDMI conectors.SchDoc


Title HDMI Transmitter Board
Description: Analog to I2S
Revision: 2
Date: 19/07/2011
Engineer: Marc
Sheet 6 of 6
File: I2S_Audio.SchDoc
Altium
SW1
MicroSwitch 8 pos
IC2
PCM1804
IC3
PCM1808
CN2
CON_4
I2S_TO_FPGA
BCK_TO_FPGA
LRCK_TO_FPGA
SCK1_TO_FPGA
I2S_DOUT_PCM1804
BCK_PCM1804
LRCK_PCM1804
SCK1_PCM1804
I2S_DOUT_PCM1808
BCK_PCM1808
LRCK_PCM1808
SCK1_PCM1808
FMT0_PCM1804
FMT1_PCM1804
S/M_PCM1804
MD0_PCM1808
MD1_PCM1808
FMT_PCM1808
AUDIO_LEFT1
AUDIO_LEFT-
AUDIO_RIGHT1
AUDIO_RIGHT-
OVFL_PCM1804
OVFR_PCM1804
RST#_PCM1804
SCK1_PCM1804
LRCK_PCM1804
BCK_PCM1804
I2S_DOUT_PCM1804
OSR0_PCM1804
OSR1_PCM1804
OSR2_PCM1804
BYPASS_PCM1804
5V
3V3
GND

Annex E

Pinout of interconnection between 2C35 Kit HDMI Transmitter board

The *Annex Table E.1: Connections between Nios 2C35 Board and HDMI Generator Board*, details the pinout of the connection between the 2C35 development kit that acts as the main board, and the HDMI Transmitter board, that acts as the slave expansion board. The table details both sides of the interconnection and is very useful to provide a reference document for a successful FPGA pin assignment.

FPGA Pin	2C35 Net Name	I/O	Conn	Pin	HDMI Gen Net Name	Signal Details
J11 – PROTO1					**CN8**	
U3pin56	proto1_RESET_n	XX	J11	1	XX	Unused
E25	proto1_io0	Out	J11	3	D7	Video Data Input 7
F24	proto1_io1	Out	J11	4	D8	Video Data Input 8
F23	proto1_io2	Out	J11	5	D9	Video Data Input 9
J21	proto1_io3	Out	J11	6	D10	Video Data Input 10
J20	proto1_io4	Out	J11	7	D11	Video Data Input 11
F25	proto1_io5	Out	J11	8	D12	Video Data Input 12
F26	proto1_io6	Out	J11	9	D13	Video Data Input 13
N18	proto1_io7	Out	J11	10	D14	Video Data Input 14
P18	proto1_io8	Out	J11	11	D15	Video Data Input 15
G23	proto1_io9	Out	J11	12	D16	Video Data Input 16
G24	proto1_io10	Out	J11	13	AUX_A	Mounted 32 and R33=XX, CLK_HDMITX=AUX_A, Test in JL37
G25	proto1_io11	Out	J11	14	D17	Video Data Input 17
G26	proto1_io12	Out	J11	15	D18	Video Data Input 18
H23	proto1_io13	Out	J11	16	D19	Video Data Input 19
H24	proto1_io14	Out	J11	17	D20	Video Data Input 20
J23	proto1_io15	Out	J11	18	D21	Video Data Input 21
J24	proto1_io16	Out	J11	21	D22	Video Data Input 22
H25	proto1_io17	Out	J11	23	D23	Video Data Input 23
H26	proto1_io18	Out	J11	25	D24	Video Data Input 24
K18	proto1_io19	Out	J11	27	D25	Video Data Input 25
K19	proto1_io20	Out	J11	28	D26	Video Data Input 26
K23	proto1_io21	Out	J11	29	D27	Video Data Input 27
K24	proto1_io22	Out	J11	31	D28	Video Data Input 28
J25	proto1_io23	Out	J11	32	D30	Video Data Input 30
J26	proto1_io24	Out	J11	33	D31	Video Data Input 31
M21	proto1_io25	Out	J11	35	D32	Video Data Input 32
T23	proto1_io26	Out	J11	36	D33	Video Data Input 33
R17	proto1_io27	Out	J11	37	D34	Video Data Input 34
K21	proto1_cardsel_n	XX	J11	38	XX	Unused
P17	proto1_io28	Out	J11	39	D35	Video Data Input 35
J12 – PROTO1					**CN9**	
Y22	proto1_io40	Out	J12	3	DSD1	DSD Channel 1 Audio Data Input
T18	proto1_io29	Out	J12	4	DSD0	DSD Channel 0 Audio Data Input
T17	proto1_io30	Out	J12	5	VSYNC	Vertical Sync ADV7511
U26	proto1_io31	Out	J12	6	HSYNC	Horizontal Sync ADV7511
R19	proto1_io32	Out	J12	7	DE	Data Enable ADV7511
T19	proto1_io33	Out	J12	8	D0	Video Data Input 0
U20	proto1_io34	Out	J12	9	D1	Video Data Input 1
U21	proto1_io35	Out	J12	10	D2	Video Data Input 2
V26	proto1_io36	Out	J12	11	D3	Video Data Input 3
V25	proto1_io37	Out	J12	12	D4	Video Data Input 4

V24	proto1_io38	Out	J12	13	D5	Video Data Input 5
V23	proto1_io39	Out	J12	14	D6	Video Data Input 6
J13 – PROTO1					**CN11**	
U2pin19	proto1_osc	XX	J13	9	XX	Unused
F21	proto1_pllclk	Out	J13	11	CLK_FROM_2C35A	Mounted R33 & R32=XX, CLK_HDMITX=CLK_FROM_2C35A, Test in JL37
N26	proto1_clkout	In	J13	13	CLK_16,384_M	16,384MHz Audio Master Clock
J16 – PROTO2					**CN7**	
U3pin57	proto2_RESET_n	XX	J16	1	XX	Unused
AE24	proto2_io0	Out	J16	3	FREE_0	Free-> Pin 2 AMP
T21	proto2_io1	Out	J16	4	FREE_1	Free-> Pin 1 AMP
V22	proto2_io2	Out	J16	5	FREE_2	Free-> Pin 4 AMP
AF23	proto2_io3	Out	J16	6	FREE_3	Free-> Pin 3 AMP
AE23	proto2_io4	Out	J16	7	FREE_4	Free-> Pin 6 AMP
AC22	proto2_io5	Out	J16	8	FREE_5	Free-> Pin 5 AMP
AB21	proto2_io6	Out	J16	9	FREE_6	Free-> Pin 8 AMP
AD23	proto2_io7	Out	J16	10	FREE_7	Free-> Pin 7 AMP / (SCK,BCK) for PCM
AD22	proto2_io8	Out	J16	11	FREE_8	Free-> Pin 10 AMP / I2S_DATA for PCM
AC21	proto2_io9	Out	J16	12	FREE_9	Free-> Pin 9 AMP
AD21	proto2_io10	Out	J16	13	FREE_10	Free-> Pin 12 AMP / MCLK for PCM
AF22	proto2_io11	Out	J16	14	FREE_11	Free-> Pin 11 AMP / LRCLK for PCM
AE22	proto2_io12	Out	J16	15	FREE_12	Free-> Pin 14 AMP --> VCC_UART
V18	proto2_io13	Out	J16	16	FREE_13	Free-> Pin 13 AMP --> GND_UART
W19	proto2_io14	Out	J16	17	FREE_14	Free-> Pin 16 AMP --> RX_UART
U17	proto2_io15	Out	J16	18	FREE_15	Free-> Pin 15 AMP --> TX_UART
U18	proto2_io16	Out	J16	21	LRCLK	I2S Audio Left Right Clock Input (LRCLK)
AF21	proto2_io17	Out	J16	23	SCLK	I2S Audio Bit Clock Input (SCK/BCK)
AE21	proto2_io18	Out	J16	25	D29	Video Data Input 29
AB20	proto2_io19	Out	J16	27	I2S3	I2S Channel 3 Audio Data Input
AC20	proto2_io20	Out	J16	28	I2S_TO_FPGA	I2S Data from FPGA/PCM to ADV7511
AF20	proto2_io21	Out	J16	29	MCLK	I2S Audio Reference Clock Input (MCK)
AE20	proto2_io22	Out	J16	31	SPDID_HDMITX	SPDIF Audio Input of ADV7511
AD19	proto2_io23	Out	J16	32	DSD_CLK	DSD Clock Input
AC19	proto2_io24	Out	J16	33	DSD5	DSD Channel 5 Audio Data Input
AA17	proto2_io25	Out	J16	35	DSD4	DSD Channel 4 Audio Data Input
AA18	proto2_io26	Out	J16	36	DSD3	DSD Channel 3 Audio Data Input
W17	proto2_io27	Out	J16	37	DSD2	DSD Channel 2 Audio Data Input
AA20	proto2_cardsel_n		J16	38	XX	Unused
V17	proto2_io28	Out	J16	39	OVFL_PCM1804	PCM1804 Overflow Left signal
J15 – PROTO2					**CN10**	
AE17	proto2_io40	In	J15	3	INT_HDMITX_1	Interruption line from ADV7511
AB18	proto2_io29	In	J15	4	SPDIF_ARC_HDMITX_1	SPDIF (ARC) Audio Output from ADV7511
AC18	proto2_io30	Bidir	J15	5	SCL_SLAVE_HDMITX_1	I2C Control ADV7511 (SCL)
AF19	proto2_io31	Bidir	J15	6	SDA_SLAVE_HDMITX_1	I2C Control ADV7511 (SDA)
AE19	proto2_io32	Out	J15	7	CEC_CLK	CEC Master Clock for ADV7511
AF18	proto2_io33		J15	8	XX	Unused
AE18	proto2_io34	Out	J15	9	OSR1_PC	PCM1804 Oversampling Ratio Bit 1

					M1804	(Max 768 Fs)
AA16	proto2_io35	Out	J15	10	OSR2_PCM1804	PCM1804 Oversampling Ratio Bit 2 (Max 768 Fs)
Y16	proto2_io36	Out	J15	11	OSR0_PCM1804	PCM1804 Oversampling Ratio Bit 0 (Max 768 Fs)
AC17	proto2_io37	Out	J15	12	BYPASS_PCM1804	PCM1804 Bypass Filter Activation Bit
AD17	proto2_io38	Out	J15	13	OVFR_PCM1804	PCM1804 Overflow Right Signal
AF17	proto2_io39	Out	J15	14	RST#_PCM1804	PCM1804 Reset
J17 – PROTO2					**CN5**	
U2pin18	proto2_osc	XX	J17	9	XX	Unused
F20	proto2_pllclk	Out	J17	11	CLK_FROM_2C35B	Not Connected. Test in JL14
AF14	proto2_clkout	In	J17	13	CLK_27_M	27MHz Video Master Clock

Annex Table E.1. Connections between Nios 2C35 Board and HDMI Generator Board

Annex F

HDMI Generator Board Cores Reference. Registers Map & I/O Signals

F.1 I²C Master Core

F.1.1 Core registers list and detailed description. Memory map

In the Annex Table F.1 all the core registers accessible through the wishbone bus from the controller have been listed. In this core the registers list is included as a reference for the device driver development.

Name	Wishbone Address	Width	Access	Brief Description
PRERlo	0x00	8	RW	Clock Prescale register low byte
PRERhi	0x01	8	RW	Clock Prescale register high byte
CTR	0x02	8	RW	Control register
TXR	0x03	8	W	Transmit data register
RXR	0x03	8	R	Receive data register
CR	0x04	8	W	Command register
SR	0x04	8	R	Status register

Annex Table F.1. Video generation core registers list

In the following tables, each register is described at bit-level, exploding the meaning of each one in a more detailed way.

Control register CTR (0x02)

Bit #	Access	Description
7	RW	EN, I²C core enable bit. '1': Core enabled. '0': Core disabled.
6	RW	IEN, I²C core interrupt enable bit. '1': Interrupts enabled. '0': Interrupts disabled.
5:0	RW	Reserved

Annex Table F.2. Control register CTR. Reset Value: 0x00

Transmit register TXR (0x03)

Bit #	Access	Description
7:1	W	Next byte to transmit via I²C
6	RW	In case of data transfer this byte is the LSB In case of the slave address transfer after the START

		condition, represent the RW bit. '1': reading from slave. '0': writing to slave.

Annex Table F.3. Transmit register TXR. Reset Value: 0x00

Receive register RXR (0x03)

Bit #	Access	Description
7:0	R	Last byte received via I^2C

Annex Table F.4. Receive register RXR. Reset Value: 0x00

Command register CR (0x04)

Bit #	Access	Description
7	W	STA, generate start or repeated START condition before the byte transfer.
6	W	STO, generate stop condition after the byte transfer.
5	W	RD, read transfer from slave
4	W	WR, write transfer to slave
3	W	ACK, when receiving data from slave, to sent ACK('0') or NACK('1)
2:1	W	Reserved
0	W	IACK, interrupt acknowledge, set to clear a pending interrupt

Annex Table F.5. Command register CR. Reset Value: 0x00

Status register SR (0x04)

Bit #	Access	Description
7	R	RxACK, received ack from slave. '1': No acknowledge received. '0': Acknowledge received.
6	R	BUSY, I2C bus busy. '1': After START condition detected. '0': After STOP condition detected.
5	R	AL, arbitration lost flag. '1': Arbitration lost
4:2	R	Reserved
1	R	TIP, transfer in progress '1': When transferring data. '0': When transfer completes.
0	R	IF, interrupt flag. Set when an interrupt is pending.

Annex Table F.6. Status register SR. Reset Value: 0x00

F.2 Video Generation Core

F.2.1 Core registers list and detailed description. Memory map.

In the Annex Table F.7 all the core registers accessible through the wishbone bus from the controller have been listed.

Name	Wishbone Address	Width	Access	Brief Description
Vactive+Vsync	0x10	32	RW	Length of Vactive and Vsync pulses in lines for desired timing
Vfront+Vback	0x11	32	RW	Length of Vfront and Vback pulses in lines for desired timing
Hactive+Hsync	0x12	32	RW	Length of Hactive and Hsync pulses in pixels for desired timing
Hfront+Hback	0x13	32	RW	Length of Hfront and Hback pulses in pixels for desired timing
Control Register	0x14	32	RW	Several other information usable for video_gen_top and cb_gen_top cores

Annex Table F.7. Video generation core registers list

In the following tables, each register is described at bit-level, exploding the meaning of each one in a more detailed way.

Vactive+Vsync register (0x10)

Bit #	Access	Description
31:28	-	Unused
27:16	RW	Length of **Vactive** pulse in video lines for desired timing. 12 bit-length, max 4095 lines.
15:12	-	Unused
11:0	RW	Length of **Vsync** pulse in video lines for desired timing. 12 bit-length, max 4095 lines.

Annex Table F.8. Vactive+Vsync. Reset Value: 0x02d00005

Reset Value: 0x02d00005, means 720 active lines and a vertical sync pulse of 5 lines, an information corresponding to the typical video timing 720p.

Vfront+Vback register (0x11)

Bit #	Access	Description
31:28	-	Unused
27:16	RW	Length of **Vfront** pulse in video lines for desired timing. 12 bit-length, max 4095 lines.
15:12	-	Unused
11:0	RW	Length of **Vback** pulse in video lines for desired timing. 12 bit-length, max 4095 lines.

Annex Table F.9. Vfront+Vback. Reset Value: 0x00050014

Reset Value: 0x00050014, means 5 front porch blanking lines and 20 back porch blanking lines, an information corresponding to the typical video timing 720p.

Hactive+Hsync register (0x12)

Bit #	Access	Description
31:28	-	Unused
27:16	RW	Length of **Hactive** pulse in video pixels for desired timing. 12 bit-length, max 4095 pixels.
15:12	-	Unused
11:0	RW	Length of **Hsync** pulse in video pixels for desired timing. 12 bit-length, max 4095 pixels.

Annex Table F.10. Hactive+Hsync. Reset Value: 0x05000028

Reset Value: 0x05000028, means 1280 active pixels and an horizontal sync pulse of 40 pixels, an information corresponding to the typical video timing 720p.

Hfront+Hback register (0x13)

Bit #	Access	Description
31:28	-	Unused
27:16	RW	Length of **Hfront** pulse in video pixels for desired timing. 12 bit-length, max 4095 pixels.
15:12	-	Unused
11:0	RW	Length of **Hback** pulse in video pixels for desired timing. 12 bit-length, max 4095 pixels.

Annex Table F.11. Hfront+Hback. Reset Value: 0x006e00dc

Reset Value: 0x006e00dc, means 110 front porch blanking pixels and 220 back porch blanking pixels, an information corresponding to the typical video timing 720p.

Control Register register (0x14)

Bit #	Access	Description
31:16	-	Unused
15:8	RW	Pattern selected in the output, 255 possible options. Only a few implemented. 0x00: 100% Horizontal Color Bar Pattern 0x01: 100% Vertical Color Bar Pattern 0x10: Full 100% White Pattern 0x20: Full 100% Black 0x30: Red Pattern 0x40: Green Pattern 0x50: Blue Pattern
7:4	-	Unused
3	RW	IntProgFlag, Interlaced or Progressive timing Flag.

		This bit is used to selecte one of this options to generate the desired timing. '1': interlaced timing '0': progressive timing
2	RW	Vpol, Vertical Polarity. This bit is used to select between active high or active low polarity for the vertical sync pulse. '1': Active High or positive (defined as P in CEA-861-E tables) '0': Active Low or negative (defined as N in CEA-861-E tables)
1	RW	H pol, Horizontal Polarity. This bit is used to select between active high or active low polarity for the horizontal sync pulse. '1': Active High or positive (defined as P in CEA-861-E tables) '0': Active Low or negative (defined as N in CEA-861-E tables)
0	-	Unused

Annex Table F.12. Control Register. Reset Value: 0x00100007

Reset Value: 0x00100007, means that the selected patternat power-up is a Full 100% White Pattern, the timing desired is generated as a progressive timing and that both vertical and horizontal sync pulses are defined as Active High. This configuration corresponds to the typical video timing 720p.

F.2.2 Core Input & Output signals description

Annex Table F.13 shows the listing of all core I/O signals with a brief description of each signal.

Signal	I/O	Description
wb_clk_i wb_rst_i arst_i wb_adr_i wb_dat_i wb_dat_o wb_we_i wb_stb_i wb_cyc_i wb_ack_o	In/Out	Wishbone interface input/output signals necessary to complete transactions in the bus. For deep information about wishbone bus and the meaning of each signal refer to the hardware architecture chapter.
pixel_clk	In	Input clock signal that is fed from any of the configurable frequencies derived from video 27MHz oscillator, and used as the pixel clock to generate the video timings waveform, and to synchronously move the clocked video data outputs.
h_sync_signal_out	Out	Horizontal sync output signal to connect directly to ADV7511 input video interface.
v_sync_signal_out	Out	Vertical sync output signal to connect directly to ADV7511 input video interface.
den_signal_out	Out	Data enable output signal to connect directly to ADV7511 input video interface.
red_out[11..0]	Out	12 bit output bus to represent Red color component, with data clocked to pixel_clk frequency.
green_out[11..0]	Out	12 bit output bus to represent Green color component, with data clocked to pixel_clk frequency.

blue_out[11..0]	Out	12 bit output bus to represent Blue color component, with data clocked to pixel_clk frequency.

Annex Table F.13. Video generation core I/O signals

F.3 Configurable Clock Dividers Core

F.3.1 Core registers list and detailed description. Memory map.

In the Annex Table F.14 all the core registers accessible through the wishbone bus from the controller have been listed.

Name	Wishbone Address	Width	Access	Brief Description
divider_reg#0	0x08	32	RW	Mux selection control and control of different synchronously generated clocks (most of them are not used)
divider_reg#1	0x09	32	RW	Mux selection control and control of different synchronously generated clocks (most of them are not used)
divider_reg#2	0x0a	32	RW	Control of different synchronously generated clocks (most of them are not used)

Annex Table F.14. Configurable clock divider core registers list

In the following tables, each register is described at bit-level, exploding the meaning of each one in a more detailed way.

divider_reg#0 (audio) register (0x08)

Bit #	Access	Description
31:19	-	Not used
18:16	RW	Programmable outputs used for clock multiplexer control. (Not connected)
15:8	RW	Configured ODD_RATIO divider for the input clock. Used to divide input clock for any integer number, odd or even, but output clock has a variable duty cycle, as output clock is at high level during one input clock cycle. (Not connected) $divided\_clk\_flank = clk\_in \frac{clk\_in}{1 + ODD\_RATIO}$
7:0	RW	Configured EVEN_RATIO divider for the input clock. Used to divide input clock for even numbers keeping a 50% duty cycle ouput clock. Clock output is used as audio_master_clk for I2S generation core. $divided\_clk\_50\_duty = clk\_in \frac{clk\_in}{2 \times (1 + EVEN\_RATIO)}$

Annex Table F.15. divider_reg#0. Reset Value: 0x00000102

Reset Value: 0x00000102, means that mux control lines value is “000”, although this control lines are not used, and that ODD_RATIO and EVEN_RATIO values are configured

respectively to 1 and 2. Only the *divided_clk_50_duty* output is used and connected as audio_master_clk, that means:

$$divided\_clk\_50\_duty = audio\_master\_clk = \frac{clk\_in}{6} = 24{,}576MHz$$

divider_reg#1 (video#1) register (0x09)

Bit #	Access	Description
31:19	-	Not used
18:16	RW	Programmable outputs used for clock multiplexer control. Connected to video clock multiplexer.
15:8	RW	Configured ODD_RATIO divider for the input clock. Used to divide input clock for any integer number, odd or even, but output clock has a variable duty cycle, as output clock is at high level during one input clock cycle. (Not connected) $divided\_clk\_flank = clk\_in \frac{clk\_in}{1 + ODD\_RATIO}$
7:0	RW	Configured EVEN_RATIO divider for the input clock. Used to divide input clock for even numbers keeping a 50% duty cycle ouput clock. (Not connected) $divided\_clk\_50\_duty = clk\_in \frac{clk\_in}{2 \times (1 + EVEN\_RATIO)}$

Annex Table F.16. divider_reg#1. Reset Value: 0x00040100

Reset Value: 0x00040100, means that mux control lines value is “100”, and that means that the fourth mux input *main_video_clk_74_25* is used as the default pixel clock. That makes sense as the 74.25MHz pixel clock corresponds to the typical video timing 720p, and all the default values for video generation core registers are also prepared for 720p video timing. ODD_RATIO and EVEN_RATIO values are configured respectively to 1 and 0, but clock outputs are not connected.

divider_reg#2 (video#2) register (0x0a)

Bit #	Access	Description
31:19	-	Not used
18:16	RW	Programmable outputs used for clock multiplexer control. Connected to video clock multiplexer.
15:8	RW	Configured ODD_RATIO divider for the input clock. Used to divide input clock for any integer number, odd or even, but output clock has a variable duty cycle, as output clock is at high level during one input clock cycle. (Not connected) $divided\_clk\_flank = clk\_in \frac{clk\_in}{1 + ODD\_RATIO}$
7:0	RW	Configured EVEN_RATIO divider for the input clock. Used to divide input clock for even numbers keeping a 50% duty cycle ouput clock. (Not connected)

		$divided\_clk\_50\_duty = clk\_in \frac{clk\_in}{2 \times (1 + EVEN\_RATIO)}$

Annex Table F.17. divider_reg#2. Reset Value: 0x00000100

Reset Value: 0x00000100, means that mux control lines value is "000", although this control lines are not used. ODD_RATIO and EVEN_RATIO values are configured respectively to 1 and 0, but clock outputs are also not connected.

F.3.2 Core Input & Output signals description

Annex Table F.18 shows the listing of all core I/O signals with a brief description of each signal.

Signal	I/O	Description
wb_clk_i wb_rst_i arst_i wb_adr_i wb_dat_i wb_dat_o wb_we_i wb_stb_i wb_cyc_i wb_ack_o	In/Out	Wishbone interface input/output signals necessary to complete transactions in the bus. For deep information about wishbone bus and the meaning of each signal refer to the hardware architecture chapter.
clk	In	Input clock used to generate divide output clocks.
clk_div_2	Out	Fixed divided by 2 clock output.
clk_div_4	Out	Fixed divided by 4 clock output.
divided_clk_50_duty	Out	Programmable clock output used to generate a clock synchronous to input clock divided by an even integer and with a 50% duty cycle.
divided_clk_flank	Out	Programmable clock output used to generate a clock synchronous to input clock divided by an odd or even integer but with a variable duty cycle.
mux_control_lines[15..0]	Out	15 bit programmable output bus. In the second instantiation of the core, the three low bits are used as select lines for video clock multiplexer..

Annex Table F.18. Configurable clock dividers block I/O signals

F.4 I²S Generation Core

F.4.1 Core generics list and detailed description.

In the Annex Table F.19 all the core generics configurable at compilation time have been listed.

Name	Type	Default	Brief Description
WISHBONE_ADDR	std_logic_vector	"000"	Configurable base address for top Wishbone slave core.
DATA_WIDTH	integer	32	Wishbone data bus width. It needs to be 32 if the samples word length has to be 32

ADDR_WIDTH	integer	12	Wishbone addresses bus width. The MSB determines if a transaction is addressed to configuration interface or sample buffer interface.
NUM_SAMPLES_LEFT	integer	1920	Num Samples Left ROM (M) default value. It must match with the ROM size and can vary between $[0,2^{16}]$
NUM_SAMPLES_RIGHT	integer	1920	Num Samples Right ROM (M) default value. It must match with the ROM size and can vary between $[0,2^{16}]$
JUMP_COUNTER_SAMPLES_LEFT	integer	1	Default value for programmable address counter, also called phase accumulator, used to skip some values while reading from LUT to generate left channel data stream.
JUMP_COUNTER_SAMPLES_RIGHT	integer	4	Default value for programmable address counter, also called phase accumulator, used to skip some values while reading from LUT to generate right channel data stream.
ROM_ADDR_LEFT_WIDTH	integer	11	Address width for left channel ROM LUT.
ROM_ADDR_RIGHT_WIDTH	integer	11	Address width for right channel ROM LUT. In current system there is only one LUT for left and right channels.
MAXMEMO	integer	1920	Maximum number of samples to completely fill the sample buffer.

Annex Table F.19. I^2S generation core generics list

F.4.2 Core registers list and detailed description. Memory map.

In the Annex Table F.20 all the core registers accessible through the wishbone bus from the controller have been listed.

Name	Wishbone Address	Width	Access	Brief Description
tx_config_wb	0x18	32	RW	This register is a mirror of the I^2S transmitter core configuration register. It is ued to configure clock rates, audio word length and interrupts behaviour.
rom_config1_wb	0x1a	32	RW	Number of samples for right and left ROMs.
rom_config2_wb	0x1b	32	RW	It allows mute control for every channel, and it is also used to enable or disable the continuous I^2S stream generation, and to configure the ROM address width.
rom_config3_wb	0x1c	32	RW	It allows to change the frequency of the audio outputs by stepping faster or slower thrpugh the sine LUT.

Annex Table F.20. I^2S generation core registers list

In the following tables, each register is described at bit-level, exploding the meaning of each one in a more detailed way.

tx_config_wb register (0x18)

Bit #	Access	Description
31:22	-	Not used
21:16	RW	RES, sample data word length. It can be configured from 16 to 32 bits, but 32 bits are only supported if wishbone data bus width DATA_WIDTH is configured to 32 bits too.
15:8	RW	RATIO, clock divider for the transmit frequency. The audio input clock is divided by a factor of 2*(2+RATIO) to generate the serial bit clock (BCK) and by a factor (2+RATIO) to generate the master clock (MCK). The sample rate is derived from the relation Fs=LRCK=BCK/2*RES
7:3	-	Not used.
2	RW	TSWAP, bit used to swap between left and right channel position in the I^2S data stream. '0': Left channel use samples stored in even adresses. '1': Left channel use samples stored in odd adresses.
1	RW	TXINTEN, interrupt output enable flag, used to enable/disable the interrupt generation to notify an empty sample buffer. '0': Interrupt output disabled. '1': Interrupt output enabled.
0	RW	TXEN, core enable flag, used to enable/disable the I^2S data stream generation. '0': Transmitter core disabled. '1': Transmitter core enabled.

Annex Table F.21. tx_config_wb. Reset Value: 0x00200403

Reset Value: 0x00200403, means that:

- RES=0x20. Sample data word length is configured to 32 bits.
- RATIO=4. Master clock, bit rate and sample rate are configured as follows:

$$MCK = \frac{main\_audio\_clk}{RATIO+2} = \frac{147{,}456MHz}{6} = 24.576MHz$$

$$BCK = \frac{main\_audio\_clk}{2\times(RATIO+2)} = \frac{147{,}456MHz}{12} = 12.288MHz$$

$$LRCK = \frac{BCK}{2\times RES} = \frac{147{,}456MHz}{64} = 192kHz$$

- TSWAP=0. Left channel stored in even addresses and sent when LRCK is low.
- TXINTEN=1. Interrupt generation enabled.
- TXEN=1. Core enabled.

rom_config1_wb register (0x1a)

Bit #	Access	Description
31:16	RW	NUM_SAMPLES_RIGHT, Num Samples Right ROM (M) configurable at execution time. It must match with the ROM size and can vary between $[0,2^{16}]$

15:0	RW	NUM_SAMPLES_LEFT, Num Samples Left ROM (M) configurable at execution time. It must match with the ROM size and can vary between [0,2^{16}]

Annex Table F.22. rom_config1_wb. Reset value: 0x07800780

Reset Value: 0x07800780. It is really loaded from core generics NUM_SAMPLES_RIGHT and NUM_SAMPLES_LEFT configurable at run time. Although the core uses the value from the register in order to be able to change it during execution time to enable different ROMs usage, this funcionality is not fully used in this simple implementation, as only one ROM is used for left and right channels, and the number of samples is fixed to 1920.

rom_config2_wb register (0x1b)

Bit #	Access	Description
31:30	-	Not used
29	RW	RIGHT_MUTE, mute enable flag for right channel. If enabled the core fills the sample buffer with zero-value samples instead of using the samples of the sine wave stored in the ROM.
28	RW	LEFT_MUTE, mute enable flag for left channel. If enabled the core fills the sample buffer with zero-value samples instead of using the samples of the sine wave stored in the ROM.
27:0	-	Not used.

Annex Table F.23. rom_config2_wb. Reset Value: 0x00000000

Reset Value: 0x07800780 is really loaded from core generics configurable at run time. Although this value is loaded from a register in order to be able to change it during execution time, this funcionality is not going to be used as only one ROM is used for left and right channels, and the number of samples is fixed to 1920.

rom_config3_wb register (0x1c)

Bit #	Access	Description
31:16	RW	JUMP_COUNTER_SAMPLES_RIGHT, the binary tuning word that detemines the phase jump size when filling the right channel side of the sample buffer. It is used to change the output frequency depending also on the configured sample rate frequency. It can vary between 1 and half the ROM size [1,960]
15:0	RW	JUMP_COUNTER_SAMPLES_LEFT, the binary tuning word that detemines the phase jump size when filling the right channel side of the sample buffer. It is used to change the output frequency depending also on the configured sample rate frequency. It can vary between 1 and half the ROM size [1,960]

Annex Table F.24. rom_config3_wb. Reset Value: 0x00040001

Reset Value: 0x00040001 is really loaded from core generics configurable at run time. It means that with a default sampling rate of 192 kHz:

- M_{left}=1, M_{right}=4
- The output frequencies for every channel are configured by default as follows:

$$f_{left} = \frac{M_{left} \times f_{in}}{2^N} = \frac{1 \times 192k}{1920} = 100Hz$$

$$f_{right} = \frac{M_{right} \times f_{in}}{2^N} = \frac{4 \times 192k}{1920} = 400Hz$$

More information about the sine wave generation and the meaning of this register values can be found in the subsection 9.4.3.3 - Sine Wave generation using DDS techniques.

F.4.3 Core Input & Output signals description

Annex Table F.25 shows the listing of all core I/O signals with a brief description of each signal.

Signal	I/O	Description
wb_clk_i wb_rst_i arst_i wb_adr_i wb_dat_i wb_dat_o wb_we_i wb_stb_i wb_cyc_i wb_ack_o	In/Out	Wishbone interface input/output signals necessary to complete transactions in the bus. For deep information about wishbone bus and the meaning of each signal refer to the hardware architecture chapter.
audio_clk_i	In	Input clock signal that is fed from the 147,456MHz audio master reference clock, synthesized with the PLL from the 16,384MHz external audio input clock. It is used in the I2s transmitter clock to derive the proper I^2S clocks.
i2s_sd_o	Out	I^2S Data Line output.
i2s_sck_o	Out	I^2S Bit Clock output.
i2s_ws_o	Out	I^2S Left/Right Word Clock output.
i2s_mck_o	Out	I^2S Master Clock output.

Annex Table F.25. I^2S generation core I/O signals

F.5 HW Interrupts & Reset Control Core

F.5.1 Core registers list and detailed description. Memory map.

In the Annex Table F.26 all the core registers accessible through the wishbone bus from the controller have been listed. The two internal registers share the same address as one has read-only access and the other write-only access, so data in the wishbone bus can be presented from interrupts register in a read transaction and registered to reset register in a write transaction. Although in the prototype is not really needed, this is a technique to compress the memory map.

Name	Wishbone Address	Width	Access	Brief Description

interrupts	0x05	32	RO	Interrupt status register.
resets_outs	0x05	32	WO	Cofigurable reset outputs register

Annex Table F.26. HW interrupts & reset control core registers list

In the following tables, each register is described at bit-level, exploding the meaning of each one in a more detailed way.

reset_outs register (0x05)

Bit #	Access	Description
31:7	W	Other controllable reset outputs (Not used)
6	W	Controllable reset signal connected to I2S Generation Core
5	W	Controllable reset signal connected to Video Generation Core
4	W	Other controllable reset outputs (Not used)
3	W	Controllable reset signal connected to the Video Clock Divider Core #2
2	W	Controllable reset signal connected to the Video Clock Divider Core #1
1	W	Controllable reset signal connected to the Audio Clock Divider Core #1
0	W	Controllable reset signal connected to I2C Master Core

Annex Table F.27. reset_outs. Reset Value: 0x00000000

interrupts register (0x05)

Bit #	Access	Description
31:9	R	Registered interrupts (Not used)
8	R	Registered external IRQ from ADV7511 HDMI Transmitter
7:1	R	Registered interrupts (Not used)
0	R	Registered internal IRQ from I2C Master Core

Annex Table F.28. interrupts. Reset Value: Read-Only

F.5.2 Core Input & Output signals description

Annex Table F.29 shows the listing of all core I/O signals with a brief description of each signal.

Signal	I/O	Description
wb_clk_i wb_rst_i arst_i wb_adr_i wb_dat_i wb_dat_o wb_we_i wb_stb_i wb_cyc_i wb_ack_o	In/Out	Wishbone interface input/output signals necessary to complete transactions in the bus. For deep information about wishbone bus and the meaning of each signal refer to the hardware architecture chapter.
external_irq	In	External interrupt from ADV7511 HDMI Transmitter.
irq_cores	In	Internal interrupts from cores.

reset_cores	Out	Reset outputs distributed to different cores in the system.

Annex Table F.29. Configurable clock dividers block I/O signals

Annex G

HDMI Generator control commands list

G.1 Filesystem commands

G.1.1 filecreate

Name
filecreate – creates an ESIIC .hex file and allocates needed bytes of memory for newly created file

Synopsis
filecreate -FILENAME -SIZE

Description
Creates FILENAME as an ESIIC .hex file with SIZE bytes, allocated but not initialized, remaining with indeterminate values. The newly created file can be managed and destroyed with other filesystem commands.
The ESIIC filesystem allows a maximum of 10 files allocated at same time.

Options

Examples
```
>> filecreate –i2cframe -7
```

Diagnostics
It returns OK if file is successfully created.
```
>> |OK|09|02|0000|File: FILENAME has been successfully created
```

Otherwise it returns NG with specific error.
```
>> |NG|09|02|0000|It is impossible to create two files with the same name
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend

Author
Marc Blanch

G.1.2 fileadd

Name
fileadd – adds to to the end of an already created ESIIC .hex file the binary data represented by the specified hexadecimal string.

Synopsis
fileadd -FILENAME –HEXADECIMAL_STRING

Description
The HEXADECIMAL_STRING data is converted to binary data and added to the end of an already created file FILENAME.
HEXADECIMAL_STRING specified has a to be a valid value from 00 to FF and HEXADECIMAL_STRING size has to be multiple of 2.
For example:
00FF0F is a valid value.
00FF0G is not valid because G does not have a representation in an hexadecimal numeral system.
00FF0 is not valid because the size of string is not multiple of 2. The last nibble would have an indeterminate value.
The end pointer is automatically updated assuring next call to fileadd appends contents from the end of the file. The fileadd command can be used unlimited number of times while the file is not full, or size of HEXADECIMAL_STRING is not greater than remaining allocated size in the file. The size of data already added must be controlled by the application running in client side.

Options

Examples
```
>> fileadd –i2cframe -0300300002337A
```

Diagnostics
It returns OK if data is succesfully added to the file.
```
>> |OK|09|02|0000|The line have been added to the file
```

Otherwise it returns NG with specific error.
```
>> |NG|09|02|0000|Overflow the size of file
>> |NG|09|02|0000|The specified file doesn't exist in NIOSII filesystem
>> |NG|09|02|0000|Format Error, Impossible to add the line to file
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend

Author
Marc Blanch

G.1.3 fileerase

Name
fileerase – erases an ESIIC .hex file making free the file slot and all memory previoulsy allocated with a call to *filecreate* is deallocated, making it available again.

Synopsis
fileerase –FILENAME

Description
Erases FILENAME which is an ESIIC .hex file previously created.
Also deallocates all the memory space used by the file.
The ESIIC filesystem allows a maximum of 10 files allocated at same time.

Options

Examples
```
>> fileerase –i2cframe
```

Diagnostics
It returns OK if file is successfully erased.
```
>> |OK|09|02|0000|The file have been removed
```

Otherwise it returns NG with specific error.
```
>> |NG|09|02|0000|The specified file doesn't exist in NIOSII filesystem
>> |NG|09|02|0000|Format error: -file parameter not found
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend

Author
Marc Blanch

G.1.4 fileempty

Name
fileempty – removes the contents of an ESIIC .hex file but does not free the file slot and all memory remains allocated.

Synopsis
fileempty –FILENAME

Description
Removes the contents of FILENAME which is an ESIIC .hex file previously created.
The end pointer is automatically updated to first position of previously allocated memory, assuring next call to fileadd would add contents from the start of the file. However the memory is not initialized, remaining with indeterminate values.
The file goes to the same state of a newly created file.

Options

Examples
```
>> fileerase –i2cframe
```

Diagnostics
It returns OK if data is successfully removed from file.
```
>> |OK|09|02|0000|The file contents have been removed
```

Otherwise it returns NG with specific error.
```
>> |NG|09|02|0000|The specified file doesn't exist in NIOSII filesystem
>> |NG|09|02|0000|Format error: -file parameter not found
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend

Author
Marc Blanch

G.1.5 filechecksum

Name
filechecksum – calculates the 16-bit checksum of an ESIIC .hex file

Synopsis
filechecksum –FILENAME

Description
Calculates the 16-bit checksum of data contained in FILENAME which is an ESIIC .hex file previously created.
The checksum 4 hex digits are calculated by adding together the hex-encoded bytes of all data in FILENAME, then leaving only the 2 LSB of the result, and making a 2's complement (inverting the two bytes and adding 0x0001).
For example, a file with the following data 0300300002337A
03 + 00 + 30 + 00 + 02 + 33 + 7A = E2, 2's complement is FF1E

Options

Examples
```
>> filechecksum -i2cframe
```

Diagnostics
It returns the read checksum of the file if calculation is succesfully done.
```
>> |RT|09|02|0002|FF1E
```

Otherwise it returns NG with specific error.
```
>> |NG|09|02|0000|The specified file doesn't exist in NIOSII filesystem
>> |NG|09|02|0000|Format error: -file parameter not found
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend

Author
Marc Blanch

G.1.6 fileupsend

Name
fileupsend – returns the contents of an ESIIC .hex file

Synopsis
fileupsend –FILENAME

Description
Returns all data contained in FILENAME which is an ESIIC .hex file previously created.

Options

Examples
```
>> fileupsend –i2cframe
```

Diagnostics
It returns the read data contents of the file.
```
>> |RT|09|02|0007|0300300002337A
```

Otherwise it returns NG with specific error.
```
>> |NG|09|02|0000|The specified file doesn't exist in NIOSII filesystem
>> |NG|09|02|0000|Format error: -file parameter not found
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend

Author
Marc Blanch

G.2 I²C Commands

G.2.1 i2c1

Name

i2c1 – allows access to the I²C master controller number #1 in order to read from slave devices or write to slave devices present in the I²C bus #1. It allows several modes of operation with different command switches that become supported or not depending on the mode of operation.

Synopsis

i2c1 **–MODE –rate** *RATE* [**–disable_core**]

Specifying the different MODE options:

i2c1 **–read –rate** *RATE* **–file** *FILE* **–i2caddr** *I2CADDR* [**–subaddrini** *SUBADDR*] [**–subaddrend** *SUBADDR*] **–num_bytes** *NUM_BYTES* [**–disable_core**]

i2c1 **–write –rate** *RATE* **–file** *FILE* [**–ticks** *TICKS_MS*]

i2c1 **–read_rx –rate** *RATE* **–i2caddr** *I2CADDR* [**–subaddrini** *SUBADDR*] **–num_bytes** *NUM_BYTES* [**–disable_core**]

i2c1 **–write_data –rate** *RATE* **–i2caddr** *I2CADDR* [**–subaddrini** *SUBADDR*] **–num_bytes** *NUM_BYTES* **–data** *DATA* [**–disable_core**]

Description

This command allows different operation with the I²C master controller. It has 4 modes of operation depending on the direction of the I²C transfer (read/write) and depending if data is got/collected to/from and ESIIC .hex file, or data is got/collected from/with the command or command response.
Different command switches are allowed depending on the mode of operation

Options

–MODE

The MODE switch defines the master controller mode of operation and the I²C bus transfer. List of allowed modes *read/write/read_rx/write_data*:
MODE=read is used to read data from a I²C slave device to an ESIIC .hex file previously created. The contents in the ESIIC .hex file are overwritten.
The ESIIC .hex file format is specifically designed to transfer the contents of the .hex file to an I²C device, for this reason in the file there are some external information not contained in typical intel .hex files, specifically all the information related to the I²C transfer of data, like the I²C device address or the part of the I²C device subaddress, information that is not included in an Intel .hex file.
MODE=write is used to send data to an I²C slave device from an ESIIC .hex file previously created.

MODE=read_rx is used to read data from a I^2C slave device with features specified in the command and return read data in the same command response. There are some mandatory command switches only used in this mode.
MODE=write_data is used to send data to an I^2C slave device with features specified in the command.and data also specified in the same command. There are some mandatory command switches only used in this mode.

Supported in all the modes:
–rate *RATE*

The RATE parameter defines the I^2C bus speed for the incoming transfer, and allows values from 10000(10Kbit/s) to 3400000(3,4Mbit/s). Common I²C bus speeds are the 3,4 Mbit/s *fast mode*, 400 Kbit/s *fast mode*, 100 Kbit/s *standard mode* and the 10 kbit/s *low-speed* mode, but arbitrarily low clock frequencies are also allowed.

–disable_core

This optional switch forces the core to be disabled after the transfer. If it is not specified the master core is not disabled after the transfer, if it is specified the core is left unconfigure in the same state after a reset.

Supported only in read/write modes:
–file *FILE*

The FILE parameter defines the previously created ESIIC .hex file that will be used to send data to an I²C device in write mode and to receive data from an I²C device in read mode. This switch is mandatory in this modes.

Supported only in read/read_rx/write_data modes
–i2caddr *I2CADDR*

The I2CADDR parameter defines the 8-bit I^2C address of the slave device. The parameter has to be specified using a 8-bit hexadecimal string [00,FF]. In some slave devices the I^2C addreess is specified with 7-bits notation, as the I^2C protocol really always talks about a 7-bit wide address space, in that cases to find the 8-bit equivalent the I^2C address must be left-shifted one bit.
The explanation is that this parameter really defines directly the data used in the first byte of the I^2C transfer after the START condition, that contains not only the slave address in the first 7 bits but also the data direction bit that is always 0 (write) after a START condition. If the I^2C transfer is a read operation. master controller automatically puts the slave direction bit to 1 (read) after the REPEATED START condition.

[**–subaddrini** *SUBADDR*]

The SUBADDR parameter defines the 8 to 32-bit I^2C subaddress of the slave device. The parameter has to be specified using a 8-bit to 32-bit zero-padded hexadecimal string [00,FFFFFFFF] and the length of the string determines the number of subaddress bytes to send upto 4 bytes.
Really this command switch has been developed to help the developer that is using this command, as I^2C protocol does not define any subaddress, anyway the typical memory devices with an I^2C bus interface, have write and random read modes where the first data send after the I^2C address byte is a dummy write and represents the memory direction of the device where data should be stored in case of a write transfer or the direction from which data is sent to the bus by the slave

in case of a read transfer. In case of a write transfer to memory device the internal memory addressing subaddress can be included as part of the data, as there is no difference from the master side between that dummy data and the real data that is going to be stored in that memory position.
This command switch is optional, and in case it is not defined in the command, no subaddress is used in the I^2C transfer.
Figure Annex G.1, Figure Annex G.2 and Figure Annex G.3 show examples of I^2C transfers taken from the datasheet of an I^2C EEPROM device to illustrate some transfers with or without internal memory addressing using a subaddress..

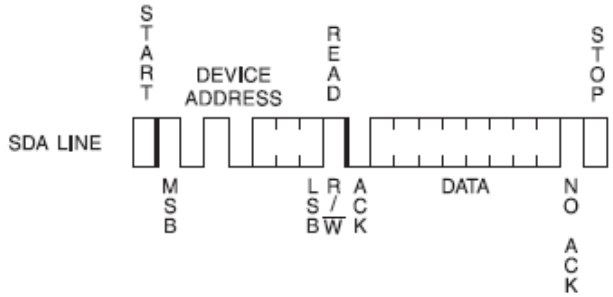


Figure Annex G.1. Current address read without subaddress to read data from last memory counter position

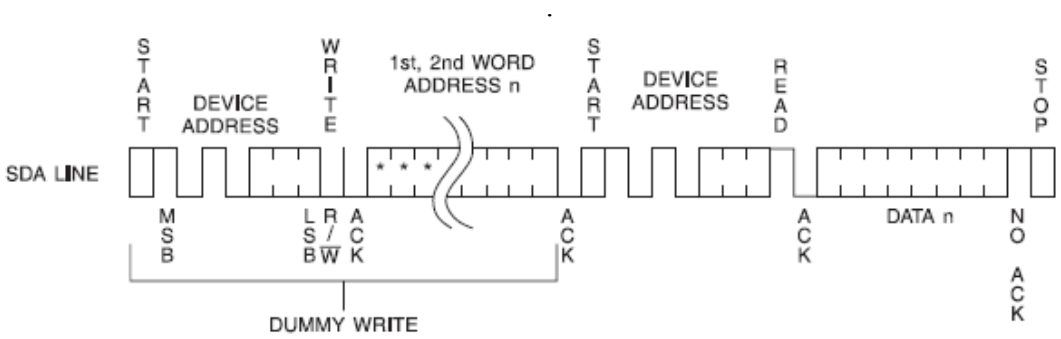


Figure Annex G.2. Random read with a specified subaddress to read data from a specific memory position

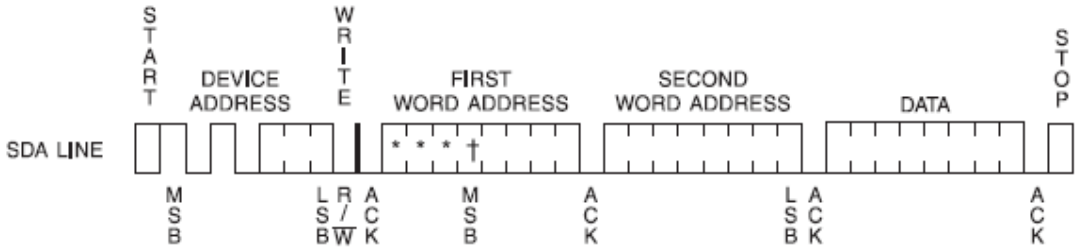


Figure Annex G.3. Byte Write to a memory position specified in the first data bytes

–num_bytes *NUM_BYTES*

The NUM_BYTES parameter defines the number of bytes to read from the I^2C slave device in a single I^2C transfer in case of read/read_rx modes and the number of bytes to write to the I^2C slave device in case of write_data mode. The NUM_BYTES parameters is specifed as a decimal base integer.
When this command switch is used in the read mode, the master can make an umlimited number of read transfers started with a START condition and finished with a STOP_CONDITION to completely fill the ESIIC .hex file until the last subaddress read specified with the –subaddrend command switch. In that case –num_bytes really means the number of byes read in a single transfer, although to read until the final subaddress, n transfers of NUM_BYTES bytes could be needed.
When this command switch is used in n the write_data mode, the data sent to the bus must be also specified in the command with the –data command switch.

Supported only in read mode
[**–subaddrend** *SUBADDR*]

The SUBADDR parameter defines the 8 to 32-bit final I^2C subaddress that is read from the I^2C slave device to a ESIIC .hex format file previously created. The parameter has to be specified using a 8-bit to 32-bit zero-padded hexadecimal string [00,FFFFFFFF] and the length of the string determines the number of subaddress bytes to send upto 4 bytes. In the case it is defined the lenght of the string has to be the same that in the **–subaddrini** *SUBADDR*
It is only used to redimensionate the length of the ESIIC .hex file.
This command switch is optional, and in case it is not defined in the command, no subaddress is used in the I^2C transfer.

Supported only in write mode
[**–ticks** *TICKS_MS*]

The *TICKS_MS* parameter defines a delay in milliseconds between a STOP condition and the next START condition in the case of a continuous transfer to an I^2C slave device from an ESIIC .hex file. In the ESIIC .hex file format, every line is interpreted as a new I2C transfer started with a START condition and finished with a STOP condition afeter all the data has been sent. This parameter is used for slow memory devices that do not support the clock stretching mechanisms to slow the data transfer from the master. This cheap devices rely in a minimum time between transfers. The parameter is optional and in case it is not specified the delay is 0 milliseconds.

Supported in write_data mode
–data *DATA*

The DATA parameter defines the data sent to an I^2C slave device.
The parameter has to be specified using a zero-padded hexadecimal string [00,FF...FF] and the length of the string is determined by the **–num_bytes** *NUM_BYTES* command switch.

Examples

1. To write the data from i2c_commands_batch ESIIC .hex file that contains the needed slave device information.

```
>> i2c1 -write -rate 100000 -file i2c_commands_batch
```

2. To read data from I2C slave device with address 0x72 to the dumped_memory ESIIC .hex file. The read operation starts from position 0x00 of the slave I^2C device and ends in position F0. All the memory dump is done with single 16 bytes I^2C read transfers, so finally 16x16 bytes are read from the I^2C device and dumped to the ESIIC .hex file.

```
>> i2c1 -read -rate 100000 -file dumped_memory -i2caddr 72 -subaddrini 00 -
subaddrend F0 -num_bytes 16
```

3. To read 1 byte of data in the memory subaddress 0xF5 from the I^2C slave device with address 0x72 and get the data in the command response.

```
>> i2c1 -read_rx -rate 100000 -i2caddr 72 -subaddrini F5 -num_bytes 1
```

4. To write 1 byte of data, 0x96, to the memory subaddress 0xAF of the I2C slave device with address 0x72..

```
>> i2c1 -write_data -rate 100000 -i2caddr 72 -subaddrini AF -num_bytes 1 -
data 16
```

Diagnostics

In case of a comand syntax error it returns NG with specific error.

```
>> |NG|15|00|000001|Incorrect Format: -num_bytes <value> not found
```

When command is correctly parsed and executed, the command response depends of the operation mode.

In write mode:

```
>> filecreate -commands -5
>> fileadd -commands -020072AF16
>> i2c1 -write -rate 100000 -file i2c_commands_batch
```

It returns OK if write transfer is correctly completed without errors.

```
>> |OK|15|00|0001|File Succesfully send via I2C bus
```

Otherwise it returns NG with specific error.

```
>> |NG|15|00|000001|File specified with argument -file <filename> doesn't
exist in NIOSII filesystem
```

In read mode:

```
>> filecreate -dumped_memory -304 -256+16+16+16
>> i2c1 -read -rate 100000 -file dumped_memory -i2caddr 72 -subaddrini 00 -
subaddrend F0 -num_bytes 16
```

It returns OK if read/write transfer is correctly completed without errors.

```
>> |OK|15|00|0001|File Succesfully read via I2C bus
```

Otherwise it returns NG with specific error.

```
>> |NG|09|02|0000|It is impossible to create two files with the same name
```

In read_rx mode:

```
>> i2c1 -read_rx -rate 100000 -i2caddr 72 -subaddrini F5 -num_bytes 1
```

It returns RT if read transfer is correctly completed without errors, and data read, 0x75 in the exanple, is returned in the last part of command response.

```
>> |RT|15|00|0001|75
```

Otherwise it returns NG with specific error.
```
>> |NG|15|00|000001|No acknowledge received
```

In write_data mode:
```
>> i2c1 -write_data -rate 100000 -i2caddr 72 -subaddrini AF -num_bytes 1 -
data 16
```

It returns OK if write transfer is correctly completed without errors.
```
>> |OK|15|00|0001|Data Succesfully send via I2C bus
```

Otherwise it returns NG with specific error.
```
>> |NG|15|00|000001|No acknowledge received
```

Bugs

See Also
filecreate, fileadd, fileerase, fileempty, filechecksum, fileupsend to operate with ESIIC .hex files

Author
Marc Blanch

G.3 HDMI Generation Control Commands

G.3.1 adv7511

Name

adv7511 – allows access to the different tasks related with the Analog Devices ADV7511 HDMI transmitter located in the Proto HDMI Generation System. It allows several modes of operation with different command switches that become supported or not depending on the mode of operation.

Synopsis

adv7511 **–MODE** [**Options**]

Specifying the different MODE options:

adv7511 **–video** [**–hparams** *HACTIVE HFRONT HSYNC HBACK*] [**–vparams** *VACTIVE VFRONT VSYNC VBACK*] [**–hpol** *POLARITY*] [**–vpol** *POLARITY*] [**–pix_clk** *CLOCK*] [**–scan** *SCAN_TYPE*] [**–num_bits** *COLOR_DEPTH*] [**–aspect_ratio** *RATIO*] [**–pattern** *PATTERN_NUMBER*]

adv7511 **–audio** [**–mute** *MUTE_CHANNELS*] [**–sampling_freq** *SAMPLING_FREQUENCY*] [**–jump_count_left** *JUMP_SIZE*] [**–jump_count_right** *JUMP_SIZE*]

adv7511 **–start_system –rate** *RATE* [**–i2caddr** *I2CADDR*] [**–get_info** *INFO_ENABLE*] [**–query_time** *CYCLE_TIME*] [**–mode** *HDMI_DVI*] [**–packet_i2caddr** *I2CADDR*] [**–edid_i2caddr** *I2CADDR*] [**–cec_i2caddr** *I2CADDR*]

adv7511 **–stop_system**

adv7511 **–set_system** [**–av_mute** *MUTE_ENABLE*]

Description

This command is used to configure the Proto HDMI Generation System. Depending on the mode of operation selected with the –mode command switch, the video generation core and the I2S audio generation core that feed the ADV7511 HDMI transmitter can be independently controlled and configured.

Options

–MODE

The MODE switch defines whick configuration task is executed. List of allowed modes *video/audio/start_system/stop_system/set_system*
MODE=video is used to configure the CEA-861E compatible video generation core, as well as the other parameters configurable in the ADV7511 HDMI transmitter that also define the final video format for the HDMI link.
All the parameters that define the video timing are completely configurable, allowing to create customized video timings with different active and blanking periods as well as different video pixel clock. The polarity of horizontal and vertical syncs, as well as the line scan between progressive and interlaced.

Finally other confiigurable parameters that define the video format are the aspect ratio, between 4:3 and 16:9 as well as the bit color depth that can be configure from 8 upto 12 bits.
MODE=audio is used to configure the I^2S generation core, as well as the other parameters configurable in the ADV7511 HDMI transmitter related to the audio input signal.
MODE=start_system is used to enable the ADV7511 HDMI transmitter system, by making the NiosII aware of the ADV7511 interruptions aand different events. Tipically the system is quickly enabled in the boot of the board, after completing all the system initalization task, in that case the command does not restart the system again as it is already up an running.
MODE=stop_system is used to disable the ADV7511 HDMI transmitter system, by killing all the related tasks that are handling the HDMI transmitter interruptions and events.
MODE=set_system is used to configure special functions of the ADV7511 HDMI transmitter system not directly related with audio or video formats.

Supported in video mode:

[**–hparams** *HACTIVE HFRONT HSYNC HBACK*]

This optional switch is used to change the number of active pixels *HACTIVE*, and the duration of the horizontal blanking period, separated in three zones, front porch pixels *HFRONT*, horizontal sync pulse pixels *HSYNC* and back porch pixels *HBACK*.
All the values are defined as a decimal integer in the range [0,4095].
After changing the parameters, the video generation core immediately starts to generate a video timing with the corresponding new values.

[**–vparams** *VACTIVE VFRONT VSYNC VBACK*]

This optional switch is used to change the number of active lines V*ACTIVE*, and the duration of the vertical blanking period, separated in three zones, front porch lines V*FRONT*, vertical sync pulse lines *VSYNC* and back porch lines *VBACK*.
All the values are defined as a decimal integer in the range [0,4095].
After changing the parameters, the video generation core immediately starts to generate a video timing with the corresponding new values.

[**–hpol** *POLARITY*]

This stands for horizontal sync pulse polarity. The value ‘0’ signifies negative polarity, where the signal stays mostly high (at a logic ‘1’) and only pulses low (to a logic ‘0’) during the sync pulse.
Likewise, the value ‘1’ signifies positive polarity, where the signal stays mostly low (at a logic ‘0’) and only pulses high (to a logic ‘1’) during the sync pulse.

[**–vpol** *POLARITY*]

This stands for vetical sync pulse polarity. The value ‘0’ signifies negative polarity, where the signal stays mostly high (at a logic ‘1’) and only pulses low (to a logic ‘0’) during the sync pulse.
Likewise, the value ‘1’ signifies positive polarity, where the signal stays mostly low (at a logic ‘0’) and only pulses high (to a logic ‘1’) during the sync pulse.

[**–pix_clk** *CLOCK*]

This switch allows to select a new pixel clock for the video timing from the range of allowed clocks that can be generated.
CLOCK=27 to use a 27MHz pixel clock.
CLOCK=54 to use a 54MHz pixel clock.

CLOCK=59_4 to use a 59.4MHz pixel clock.
CLOCK=74_25 to use a 74,25MHz pixel clock.
CLOCK=148_5 to use a 148,5MHz pixel clock.
CLOCK=297 to use a 297MHz pixel clock.

[**–scan** *SCAN_TYPE*]

The *SCAN_TYPE* parameter allows to select an interlaced scanning, when *SCAN_TYPE*=interlaced, or a progressive scanning, when *SCAN_TYPE*=progressive, for the generated video timing.

[**–num_bits** *COLOR_DEPTH*]

The *COLOR_DEPTH* parameter is used to configure the ADV7511 HDMI transmiiter color depth for the generated video format, that indicates how many bits are used for color component, allowing from normal 8-bit color depth, *COLOR_DEPTH*=8*,* to 10-bit or 12-bit deep color modes, *COLOR_DEPTH=10* or *COLOR_DEPTH=12.* This switch does not change the video data outputs color depth for the video generation core, that is fixed and always uses 12 bits for color componrnt, but it forces the ADV7511 to discard some video data inputs in some cases.

[**–aspect_ratio** *RATIO*]

The *RATIO* parameter is used to configure the ADV7511 HDMI transmiiter aspect ratio for the generated video format. That is useful for the Video Identification Codes (VIC) that allow two different aspect ratio, 4:3, RATIO=4_3 and 16:9 RATIO=16_9.

[**–pattern** *PATTERN_NUMBER*]

The *PATTERN_NUMBER* parameter is used to select one of the internal test patterns. The parameter has to be specified using a 8-bit hexadecimal string [00,FF]. Some typical values:
PATTERN_NUMBER=00, 100% Horizontal Color Bar Pattern
PATTERN_NUMBER=01, 100% Vertical Color Bar Pattern
PATTERN_NUMBER=10, Full 100% White Pattern
PATTERN_NUMBER=20, Full 100% Black
PATTERN_NUMBER=30, Red Pattern
PATTERN_NUMBER=40, Green Pattern
PATTERN_NUMBER=50, Blue Pattern

Supported only in audio mode:

[**–mute** *MUTE_CHANNELS*]

This command switch is used to mute one or both, of the two audio channels transmitted in the I^2S stream generated by the I^2S generation core. There is a small list of selectable values for the *MUTE_CHANNELS* parameter:
MUTE_CHANNELS=right, to mute only right channel.
MUTE_CHANNELS=left, to mute only left channel.
MUTE_CHANNELS=all, to mute both channe,s.
MUTE_CHANNELS=no_mute, to unmute all the channels and restore normal operation to take data from ROM sine wave.

[**–sampling_freq** *SAMPLING_FREQUENCY*]

This command switch is used to change the operating sampling frequency used in the I^2S generation core, and it also configures the ADV7511 appropiate registers to corrctly detect this stream and send it in audio packets by the HDMI link. There is a small list of selectable values for the *SAMPLING_FREQUENCY* parameter:

SAMPLING_ FREQUENCY =192k, to use a192kHz sampling rate, that it is not commonly accepted for all the HDMI sinks.
SAMPLING_ FREQUENCY =96k, to use a 96kHz sampling rate.
SAMPLING_ FREQUENCY =48k, to use a 48kHz sampling rate. This is the default sample rate when the HDMI system is started jsut after board boot-up.
SAMPLING_ FREQUENCY =32k, to use a 32kHz sampling rate.
SAMPLING_ FREQUENCY =16k, to use a 16kHz sampling rate.
SAMPLING_ FREQUENCY =8k, to use a 8kHz sampling rate.

[**–jump_count_left** *JUMP_SIZE*]

This command switch is used to change the step size when reading samples of the digitzed sine wave from the ROM for the audio left channel. Depending on the length of this step, the core completes a full cycle of the sine wave stored in the ROM at different speed, and it allows to generate audio tones with different output frequencies. The ouput frequency is a combination of the samplig rate and the jump size. The *JUMP_SIZE* parameter can change from 0 to 960, that is half of the ROM samples as this maximum value is the last one allowed before aliasing occurs in the output signal. The relationship can be found with the formula:

$$f_{out} = \frac{M \times f_{in}}{L} = \frac{JUMP\_SIZE \times SAMPLING\_FREQUENCY}{1920}$$

Some common values for the *JUMP_SIZE* parameter:

JUMP_SIZE=16 means $f_{out} = 400Hz$ if *SAMPLING_ FREQUENCY* =48k.

JUMP_SIZE=40 means $f_{out} = 1kHz$ if *SAMPLING_ FREQUENCY* =48k.

JUMP_SIZE=1 means $f_{out} = 100Hz$ if *SAMPLING_ FREQUENCY* =192k.

JUMP_SIZE=4 means $f_{out} = 400Hz$ if *SAMPLING_ FREQUENCY* =192k.

JUMP_SIZE=10 means $f_{out} = 1kHz$ if *SAMPLING_ FREQUENCY* =192k.

[**–jump_count_right** *JUMP_SIZE*]

This command switch is used to change the step size when reading samples of the digitzed sine wave from the ROM for the audio right channel. It is exactly the same behaviour that for **–jump_count_left** command switch.

Supported only in start_system mode:

–rate *RATE*

The RATE parameter defines the I^2C bus speed for the communication with the ADV7511 I^2C configuration interface, and allows values from 10000(10Kbit/s) to 3400000(3,4Mbit/s). This is the maximum speed supported for the I^2C Master core, however, the maximum rate supported for the ADV7511 slave interface is 400 Kbit/s *fast mode* and usually the typical operation is at 100 Kbit/s *standard mode*. When the embedded system is started with this I^2C rate, all the read/write transfers between the master controller and the ADV7511 are made at this rate until the stand-alone system is manually stopped..

[**–i2caddr** *I2CADDR*]

The ADV7511 uses four I^2C register maps, and the address for the Main Register Maps is 0x72 or 0x7A based on the start of the PD/AD pin during power-up. In normal operation the HDMI system is started in the HDMI generation board boot-up, and during this process, both I^2C address are tested in order to see which is the correct I^2C address to acces to the ADV7511. This command switch is only used to force a system start with a specific I^2C address for the Main Register Map.

[**–mode** *HDMI_DVI*]

This command switch is used to decide if ADV7511 is used in HDMI or DVI mode. In DVI mode no packets will be sent, and all registers related to packets will be disregarded, but the default mode after boot-up is HDMI mode. In case the switch is not specified the system is put in HDMI mode.
HDMI_DVI=hdmi, for HDMI mode.
HDMI_DVI=dvi, for DVI mode.

[**–get_info** *INFO_ENABLE*]

This command switch is for debugging purposes, and when it is enabled, *INFO_ENABLE*=1, the system cyclically puts information on an available socket about the status of the ADV7511, HPD and RxSense interruptions detection, video timing detected from the video input stream, I^2S format detected, etc.

[**–query_time** *CYCLE_TIME*]

This command defines the frequency of the task that gets information about the ADV7511 status. The *CYCLE_TIME* parameter time unit is in seconds.

[**–packet_i2caddr** *I2CADDR*]

This command switch is used to change the default I^2C address for the Packet Memory Map, a mirror of the information contained in the HDMI control packets. The Packet Memory address is programmable and controlled by aregister of the Main Register Map.

[**–edid_i2caddr** *I2CADDR*]

This command switch is used to change the default I^2C address for the EDID Memory Map, a mirror of the HDMI sink EDID. The EDID Memory address is programmable and controlled by aregister of the Main Register Map.

[**–cec_i2caddr** *I2CADDR*]

This command switch is used to change the default I^2C address for the CEC Memory Map, with specific registers for CEC control. The CEC Memory address is programmable and controlled by aregister of the Main Register Map.

Supported only in set_system mode:

[**–av_mute** *MUTE_ENABLE*]

This command switch is used to enable the AV mute feature of the ADV7511 transmitter, that is very useful when changing some properties of the video and audio input data streams, because it sends a blanking signal to the HDMI sink, avoiding some video or audio artifacts when changing properties of video and audio input data streams.

Examples

1. To change current video timing to 1080p 60Hz, with 12 bits color depth and 16:9 aspect ratio

```
>> adv7511 -video -hparams 1920 88 44 148 -vparams 1080 4 5 36 -hpol 1 -
vpol 1 -pix_clk 148_5 –scan progressive -aspect_ratio 1 -num_bits 12
```

2. To change current video timing to 1080i 60Hz, and keep untouched all other settings for video format like color depth and aspect ratio.

```
>> adv7511 -video -hparams 1920 528 44 148 -vparams 540 2 5 15 -hpol 1 -
vpol 1 -pix_clk 74_25 -scan interlaced
```

3. To change current pattern to color 100% Color Bar Pattern and keep all video format settings unchanged.

```
>> adv7511 -video -pattern 00
```

4. To change the sampling frequency in the I2S generation core and the ADV7511 to 48KHz, and generate a 400Hz audio tone for left channel and a 1kHz audio tone for right channel.

```
>> adv7511 -audio -sampling_freq 48k -mute no_mute -jump_count_left 16 -
jump_count_right 40
```

5. To stop the stand-alone HDMI system based on ADV7511.

```
>> adv7511 -stop_system
```

6. To set the I2C speed between main master controller and ADV7511 to 100kHz and start the stand-alone HDMI system previously stopped.

```
>> adv7511 -start_system -rate 100000 -i2caddr 72
```

Diagnostics

In case of a comand syntax error it returns NG with specific error.

```
>> |NG|12|00|000000|Incorrect Format: value in -pix_clk <value> not correct
```

When command is correctly parsed and executed, the command response depends of the operation mode.

In video mode:

```
>> adv7511 -video -hparams 1920 88 44 148 -vparams 1080 4 5 36 -hpol 1 -
vpol 1 -pix_clk 148_5 –scan progressive -aspect_ratio 1 -num_bits 12
```

It returns always OK when configuration is succesfully completed .As the command only involves core configuration by on-chip wishbone bus, no errors are possible apart from command syntax errors.

```
>> |OK|12|00|0000|New video timings succesfully loaded
```

In audio mode:

```
>> adv7511 -audio -sampling_freq 48k -mute no_mute -jump_count_left 16 -
jump_count_right 40
```

It returns always OK when configuration is succesfully completed .As the command only involves core configuration by on-chip wishbone bus, no errors are possible apart from command syntax errors.

```
>> |OK|12|00|0000|New audio settings succesfully configured
```

In stop_system mode:

```
>> adv7511 -stop_system
```

It returns always OK when system is succesfully disabled. No errors are possible apart from command syntax errors.

```
 >> |OK|12|00|0000|System Stopped
```

In start_system mode:

```
>> adv7511 -start_system -rate 100000 -i2caddr 72
```

It returns always OK when stand-alone system is succesfully started, or it was already started during board boot-up previously. No errors are possible apart from command syntax errors..

```
>> |OK|12|00|0000|System already Started & Running
>> |OK|12|00|0000|System Started & Running
```

Bugs

See Also

i2c1, to operate directly with the ADV7511 I^2C Main Register Map, being able to change some of the settings of adv7511 command, something that is very useful when debugging and during ADV7511 function drivers development.

Author

Marc Blanch

Annex H

EthernetControl Class Reference

This annex contains the member function documentation automatically generated from the EthernetControl class library with the Doxygen tool to generate documentation from source code..

bool EthernetControl.EthernetControl.CloseConnectionWithEthBoard (out string *errorMsg*)

Generic method to close the active socket connection and release related resources.

Parameters:

errorMsg	Error message returned by the method

Returns:

PASS:1 / FAIL:0

bool EthernetControl.EthernetControl.ConnectWithEthBoard (out string *errorMsg*, string *ipString*, int *ipPort*, bool *flushMenu*, int *timeout_ms*)

Generic method to establish a connection to a socket server running at the selected IP address and attached to the selected port. Once the socket connection is established, other methods can use the socket to send configuration commands to the remote control socket server. If socket server sends menu information once the connection is established, it can optionally flushed to not interfere the normal operation.

Parameters:

errorMsg	Error message returned by the method
ipString	Socket server IP address
ipPort	Socket server port number
flushMenu	Boolean flag to flush or not flush the menu send when a connection is established
timeout_ms	Default receive and send timeout for every operation with the active socket. It can be override in some methods

Returns:

PASS:1 / FAIL:0

bool EthernetControl.EthernetControl.ReadI2cIndividualBytesEthBoard (out string *errorMsg*, string *i2c_selected*, int *i2c_rate*, byte *i2c_address*, int *subaddr_ini*, int *bytes_subaddr*, int *bytes_to_read*, out byte[] *recv_data*)

Method used to read data from an I2C slave device.

Parameters:

errorMsg	Error message returned by the method
i2c_selected	I2C master core selected: i2c1/.../i2cX
i2c_rate	I2C bus speed
i2c_address	I2C slave address

subaddr_ini	I2C memory-mapped devices subaddress
bytes_subaddr	I2C memory-mapped device address width in bytes
bytes_to_read	Number of data bytes to read
recv_data	Data array with bytes read by I2C from the memory-mapped device

Returns:

PASS:1 / FAIL:0

bool EthernetControl.EthernetControl.SendI2cIndividualBytesEthBoard (out string *errorMsg*, string *i2c_selected*, int *i2c_rate*, byte *i2c_address*, int *subaddr_ini*, int *bytes_subaddr*, int *bytes_to_send*, byte[] *tx_data*, int *recv_timeout_ms*, ref long *real_elapsed_time*)

Method used to send data to an I2C slave device.

Parameters:

errorMsg	Error message returned by the method
i2c_selected	I2C master core selected: i2c1/.../i2cX
i2c_rate	I2C bus speed
i2c_address	I2C slave address
subaddr_ini	I2C memory-mapped devices subaddress
bytes_subaddr	I2C memory-mapped device address width in bytes
bytes_to_send	Number of data bytes to send
tx_data	Data array to send by I2C to memory-mapped device
recv_timeout_ms	Socket receive timeout. Configurable to change timeout for big data or low I2C bus speed
real_elapsed_time	Command execution time (in milliseconds)

Returns:

PASS:1 / FAIL:0

bool EthernetControl.EthernetControl.SetHdmiAudioSetupEthBoardTS (out string *errorMsg*, SamplingFrequency *SamplingFreqMode*, bool *SamplingFreqDefined*, MuteCondition *MuteChannels*, bool *MuteDefined*, double *LeftChannelFrequency*, bool *JumpCountLeftDefined*, double *RightChannelFrequency*, bool *JumpCountRightDefined*, int *recv_timeout_ms*, ref long *real_elapsed_time*)

Method to configure audio setup in HDMI generator board.

Parameters:

errorMsg	Error message returned by the method
SamplingFreqMode	Sampling frequency of I2S generation core
SamplingFreqDefined	Boolean to define if sampling frequency configuration is send to the board
MuteChannels	Enable/disable audio mute in one or both channels
MuteDefined	Boolean to define if audio mute configuration is send to the board
LeftChannelFrequency	Left channel single tone frequency

JumpCountLeftDefined	Boolean to define if left channel output frequency configuration is send to the board
RightChannelFrequency	Right channel single tone frequency
JumpCountRightDefined	Boolean to define if right channel output frequency configuration is send to the board
recv_timeout_ms	Socket receive timeout to override default setting. Configurable to change timeout for slow command responses
real_elapsed_time	Command execution time (in milliseconds)

Returns:

PASS:1 / FAIL:0

bool EthernetControl.EthernetControl.SetHdmiAudioSetupEthBoardTS (out string *errorMsg*, SamplingFrequency *SamplingFreqMode*, bool *SamplingFreqDefined*, MuteCondition *MuteChannels*, bool *MuteDefined*, int *JumpCountLeft*, bool *JumpCountLeftDefined*, int *JumpCountRight*, bool *JumpCountRightDefined*, int *recv_timeout_ms*, ref long *real_elapsed_time*)

Method to configure audio setup in HDMI generator board.

Parameters:

errorMsg	Error message returned by the method
SamplingFreqMode	Sampling frequency of I2S generation core
SamplingFreqDefined	Boolean to define if sampling frequency configuration is send to the board
MuteChannels	Enable/disable audio mute in one or both channels
MuteDefined	Boolean to define if audio mute configuration is send to the board
JumpCountLeft	Left channel ROM JUMP SIZE
JumpCountLeftDefined	Boolean to define if JUMP SIZE LEFT configuration is send to the board
JumpCountRight	Right channel ROM JUMP SIZE
JumpCountRightDefined	Boolean to define if JUMP SIZE RIGHT configuration is send to the board
recv_timeout_ms	Socket receive timeout to override default setting. Configurable to change timeout for slow command responses
real_elapsed_time	Command execution time (in milliseconds)

Returns:

PASS:1 / FAIL:0

bool EthernetControl.EthernetControl.SetHdmiVideoSetupEthBoardTS (out string *errorMsg*, int *Hactive*, int *Hfront*, int *Hsync*, int *Hback*, bool *HparamsDefined*, int *Vactive*, int *Vfront*, int *Vsync*, int *Vback*, bool *VparamsDefined*, int *Hpol*, int *Vpol*, bool *bSyncPolarityDefined*, PixelClk *pixClk*, bool *bPixClkDefined*, ScanType *scanType*, bool *bScanTypeDefined*, int *numBitsColor*, bool *bColorDepthChange*, AspectRatio *aspectRatio*, bool *bAspectRatioChange*, int *patternNumber*, bool *bPatternChange*, int *recv_timeout_ms*, ref long *real_elapsed_time*)

Method to configure video setup in HDMI generator board.

Parameters:

errorMsg	Error message returned by the method
Hactive	Number of active pixels
Hfront	Length of horizontal front porch in pixels
Hsync	Length of horizontal sync pulse in pixels
Hback	Length of horizontal back porch in pixels
HparamsDefined	Boolean to define if HSYNC signal configuration is send to the board
Vactive	Number of active lines
Vfront	Length of vertical front porch lines
Vsync	Length of vertical sync pulse in lines
Vback	Length of vertical back porch in lines
VparamsDefined	Boolean to define if VSYNC signal configuration is send to the board
Hpol	HSYNC signal polarity
Vpol	VSYNC signal polarity
bSyncPolarityDefined	Boolean to define if VSYNC/HSYNC polarity configuration is send to the board
pixClk	Pixel clock frequency
bPixClkDefined	Boolean to define if pixel clock frequency configuration is send to the board
scanType	Interlaced/Progressive scan selection
bScanTypeDefined	Boolean to define if interlaced/progressive scan configuration is send to the board
numBitsColor	Color depth, number of bits per color between 8/10/12 bits
bColorDepthChange	Boolean to define if color depth configuration is send to the board
aspectRatio	Aspect ratio information between 4:3 / 16:9
bAspectRatioChange	Boolean to define if aspect ratio configuration is send to the board
patternNumber	Number of internal video pattern
bPatternChange	Boolean to define if selected pattern configuration is send to the board
recv_timeout_ms	Socket receive timeout. Configurable to change timeout for slow command responses
real_elapsed_time	Command execution time (in milliseconds)

Returns:

PASS:1 / FAIL:0

Annex I

TestStand Functional Test Sequence architecture

I.1 TestStand Sequential Model Flowchart

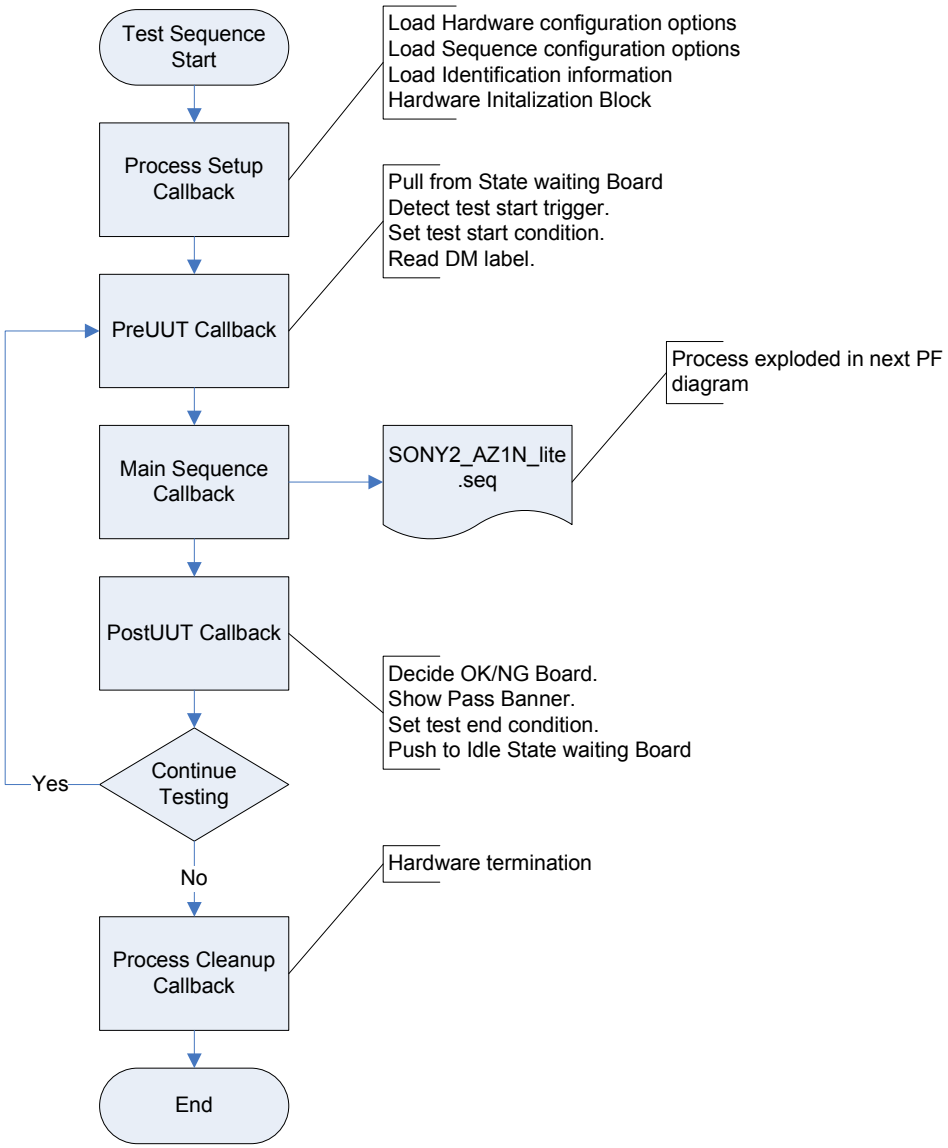


Figure Annex I.1. TestStand sequential process model flowchart for continuous testing

I.2 Test Sequence Flowchart

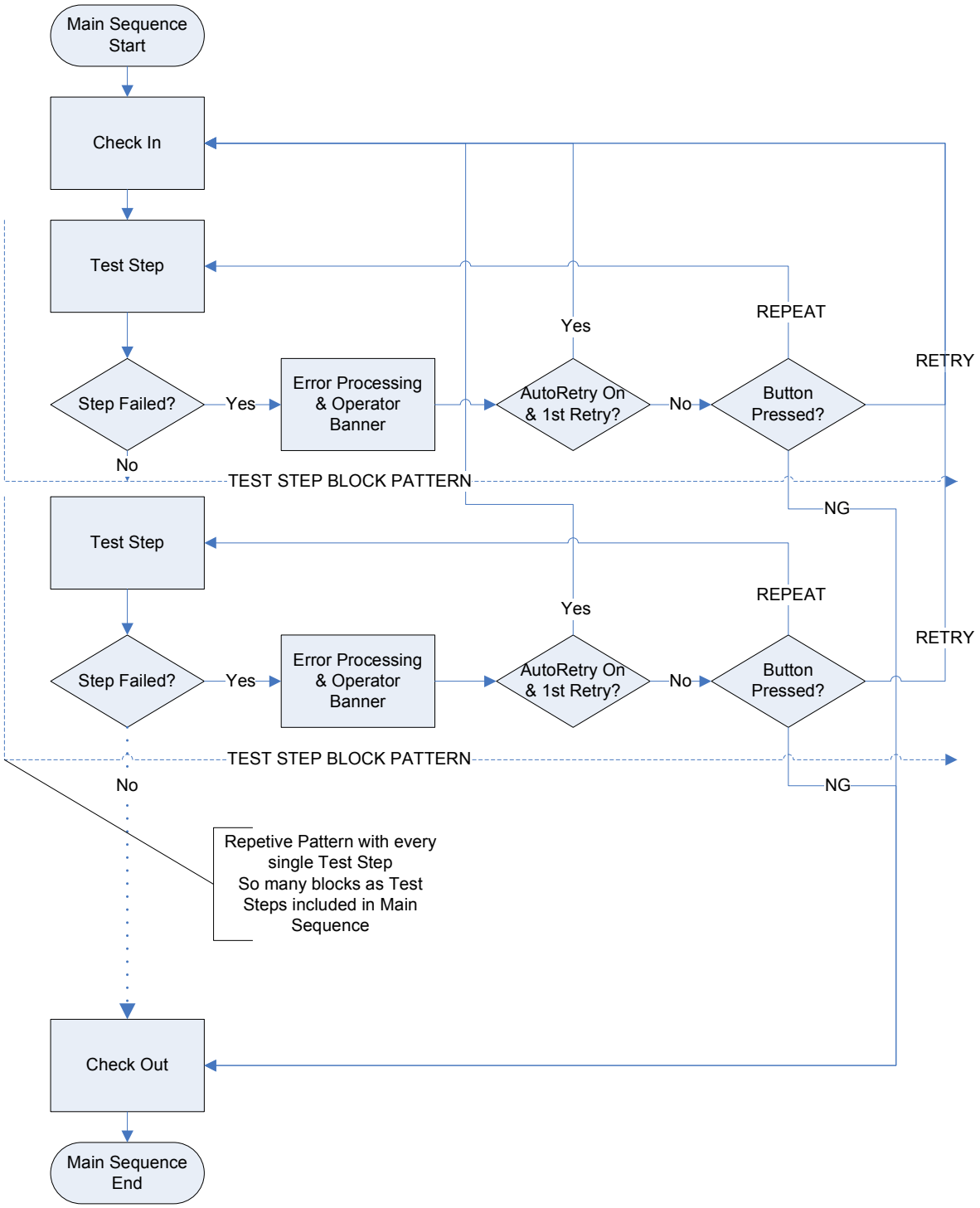


Figure Annex I.2. Test sequence for LCD TV main board flowchart

Glossary of terms

Active Line: a video line occurring during the Vactive period(s) containing both blanking and active pixels.

Application Specific Integrated Circuit (ASIC): An integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use.

Audio-Return Channel (ARC): is a new feature introduced in HDMI specification 1.4 that enables a sink device to send using the HDMI cable an upstream audio data channel in S/PDIF data format to the HDMI sink, eliminating the need of any separate audio connection between source and sink.

Automatic Optical Inspection (AOI): automatic inspection technique that uses an imaging device and an image processing software to analyze solder joints, component placement, solder paste placement, etc.

Auxiliary Video Information (AVI): additional information (defined in CEA-861-E specification) related to the video being sent from a source to a sink. In the HDMI interface is transmitted using the AVI InfoFrame packets (See InfoFrame).

Ball Grid Array (BGA): is a type of packaging used for SMT integrated circuits (IC) that consist on a grid of balls on the bottom of the package.

Bill of Materials (BOM): a Bill of Materials (BOM) is a list of the raw materials, sub-assemblies, sub-components, components, parts and the quantities of each one needed to manufacture an end product. A BOM list is always structured in a hierarchical structure, where every sub-assembly contains itself his own BOM until reaching the minimum part. In typical PCB boards, the BOM contains raw materials as PCB blank board, all electronic components of the board, from passive components to microcontrollers, and also sub-assemblies as daughter cards or generic electronic modules.

Blanking Line: a video line occurring during Vblank period(s) containing only blank pixels.

Circuit Board Adjustment (CBA): acronym used to identify the automatic functional test systems in BCN Tec factory. The document issued by designer of the electronic product explaining all the functional tests required at board-level is also known as CBA.

Clock Skew: is a problem in synchronous circuits because of variations in clock arrival times to different components.

Color Depth: the number of bits used to represent a color component sample.

Consumer Electronics Control (CEC): is a bi-directional one-wire bus that provides high-level control functions between a audiovisual products connected in a network topology by HDMI.

Context Switch: the process of saving the needed registers in the current execution on a hardware interrupt (See Interrupt Request) and restoring them on return from the interrupt service routine (See Interrupt Service Routine).

Deep Color Mode: is a term used to describe a color gamut when the system uses more than 10 bits per color component in RGB or YCbCr color spaces, allowing a color gamut of more than a billion of colors. HDMI 1.3 specification introduces deep color modes, with 30-, 36- and 48-bit RGB color depths (See Color Depth).

Device Under Test (DUT): In general electronics testing, this term refers to any electronic assembly under test, either at board-level or final system-level.

Direct Digital Synthesis (DDS): is a technique used to generate a frequency- and phase-tunable output signal referenced to a fixed frequency clock source, using a LUT (See Look-Up Table) that contains digital data of a periodic waveform.

Display Data Channel (DDC): is a standard I2C bus communication channel used for configuration and status exchange between a connected HDMI source and sink.

Dual Aspect Ratio Timing: a video format timing (See Video Timing) (e.g., 480p@50Hz) that is available in both picture aspect ratios (16:9 and 4:3) with no difference in the timing for the two formats, but it has two different Video Identification Codes (VIC) depending on the picture aspect ratio.

Enhanced Extended Display Identification Data (E-EDID): is an internal non-volatile memory structure present in the HDMI sinks, which the HDMI source reads to check the sink audio/video capabilities and general information about the sink.

Final Assy/Assembly (FA): is an acronym used to identify the manufacturing lines where the final electronic product is assembled and tested in BCN Tec factory.

Functional Test (FCT): it is a test technique based on the emulation of the final condition of the electronic product, to check if the functionalities used by the end costumer are working as expected.

HAL (Hardware Abstraction Layer): a lightweight runtime environment that provides a simple device driver interface for programs to communicate with the underlying hardware.

Hand Mount Technology (HM): is the technique or manufacturing process used to mount electronic products when the thru-hole components with wire leads, are fitted into holes of the printed circuit boards (PCBs).

High Definition (HD) Video Format: is defined a video format that has more than 720 active lines (Vactive) lines per video frame. Although is typically used only for video formats with 1080 or more active lines.

High-Bandwidth Digital Content Protection (HDCP): is a content protection standard used to prevent copying of HD digital video and audio data when it travels across compliant devices.

High-Definition Multimedia Interface (HDMI): is the first and most expanded in the industry, uncompressed, all-digital, audio/video interface alternative to typical consumer analog standards.

Hot-Plug Detect (HPD): is a pin on the HDMI connector that allows source devices to detect if a working sink is connected and ready.

In-Circuit Test (ICT): it is a test technique based on the access to nodes of the electronic circuit tested, to check for shorts, opens, resistance, capacitance or any other analog parameter to decide if circuit works appropriately or not.

InfoFrame: a data transfer structure for sending miscellaneous information from a source to a sink over an HDMI link using different packets.

Inter-IC Sound (I^2S): is a serial bus interface used for the digital transmission of audio signals encoded in a PCM stream between audio processing devices.

Inter-Integrated Circuit (I^2C): is a bi-directional two-wire serial bus that provides a communication interface between integrated circuits (ICs).

Interrupt Request (IRQ): an exception causes by a dedicated hardware request signal from an external device connected to the main processor. Typically it triggers an execution flow change to an ISR (See Interrupt Service Routine), to ensure that hardware condition is handled as quickly as possible in a deterministic way.

Interrupt Service Routine (ISR): a software routine executed to handle an interrupt request (See Interrupt Request).

Joint Test Action Group (JTAG) Boundary Scan: in electronic testing JTAG boundary scan is a technique to test interconnections inside a component that supports it. The component that supports JTAG boundary scan, has the possibility to override the functionality of each pin with the test cells programmable via the JTAG chain. That behavior can be used to construct test applications by driving signals in some pins and reading values in others to check connectivity.

Lighweight IP (lwIP): is a widely used open source TCP/IP stack designed for embedded systems.

Look-Up Table (LUT): is a data structure, commonly used to replace an expensive computation task for retrieving a value from memory. The table is populated typically with pre-calculated values and is stored in non-volatile memory.

Metastability: is a problem in electronic circuits when the output of a device is unpredictable. This state is known as metastable state. Metastability can occur when input signals to a Flip-Flop device appear asynchronously to the incoming clock domain, either for a cross-clock domain situation, excessive clock skew (See Clock Skew) or a really asynchronous signal, for example a human interaction like a pushed button.

MicroC/OS-II (uC/OS-II): is a completely portable preemptive real-time kernel (see RTOS) with the following features; multitasking, interrupt management and a lot of services such as semaphores, message queues, memory partitions, task management, time management, etc.

Minimum Quantity Order (MOQ): typically used to set the minimum quantity of an electronic component served by a manufacturer.

Multitasking: is the process of switching the central processing unit (CPU) between several tasks, sharing the CPU and maximizing its usage, thus providing a framework for modular construction of the application software.

NiosII Embedded Development Suite (EDS): the complete software environment required to build and debug software applications for the NiosII processor. The EDS includes the NiosII IDE, (See NiosII IDE) and NiosII Software Build Tools for Eclipse, (See NiosII Software Build Tools for Eclipse).

NiosII IDE: an Eclipse-based development environment for NiosII embedded designs with limited control over the software build process. The NiosII IDE is shipped as a legacy tool.

NiosII Software Build Tools (SBT) for Eclipse: an Eclipse-based development environment for NiosII embedded designs, using the SBT for project creation and detailed control over the software build process. The SBT for Eclipse provides software project management, build, and debugging capabilities.

No Failure Found (NFF): it is the situation where a failure reported by the functional test system cannot be finally found in the electronic device; typically the failure is because of a bad connection between the functional test system and the DUT.

Non-recurring engineering (NRE): Non-recurring engineering (NRE) refers to the one-time cost of researching, developing, designing, and testing a new product. It can be considered like fixed cost of the project in economic terms, differentiating from production costs that must be paid periodically during production until the product end of life. NRE costs include from wages of engineers involved in the development of the new product to new equipment that must be acquired to design or later produce this product. To assure the success of a project the NRE costs have to be kept low enough to be recovered with the future project gains.

Off-the-shelf: Off-the-shelf is a term used commonly to define technology which is ready-made and available for sale, lease, or license to a lot of possible end customers. The term can be used to refer from computer software to hardware systems like electronic integrated circuits.

Picture Aspect Ratio: ratio of width to height dimension in the active video as delivered across the uncompressed digital interface.

Pre-emption: the process of a high-priority task taking control of the cpu when a low-priority task is running. Typically it happens because the low-priority task is interrupted and loses its execution when the IRQ (See Interrupt Request) is served.

Printed Circuit Board (PCB): In the more restricting meaning, the PCB is the support where electronic components are mounted and electrically connected using signal traces etched from copper sheets “printed” onto any dielectric substrate. It is also used to refer the final condition when the support is already populated with mounted components.

Programmable Logic Device (PLD): Programmable Logic Devices are electronic components without a specific fixed function, that the final customer, rather than the chip manufacturer, programs or reconfigures to create a defined function, giving flexibility to designs with this ability to reconfigure the final logic function when desired. Depending on architecture and complexity the PLD can be classified from Complex Programmable Logic Devices (CPLDs), for small designs like combinational logic applications, to Field-Programmable Gate Arrays (FPGAs), for larger and complex designs like big state machines or soft-core microprocessors.

Quality Function Deployment (QFD) Chart: is a six-sigma tool to analyze objectively the customer needs and how different products cope with them. It is an objective method, because the rates of importance are quantified, and different options get a score depending on the priorities given when weighing the analyzed factors.

Real Time Operating System (RTOS): is an operating system intended to serve real-time application requests. It provides a nice framework to organize different features of an embedded system through a set of useful functionalities; multitasking, context switching, scheduling, intertask communication, etc.

Scheduling: this term refers to the way different tasks are assigned the CPU for execution, since in a RTOS (See RTOS) there are typically many more tasks running that need to share one or several CPUs. The assignment is carried by the RTOS scheduler.

Soft-Core Processor: a soft-core processor is an embedded microprocessor fully described in software, usually in an HDL language, which can be synthesized in programmable hardware, such as FPGAs. The main advantage of Soft-core processors implemented in FPGAs it that can be easily customized to the needs of a specific target application.

Sony/Phillips Digital Interface (S/PDIF): is a digital audio format and protocol used to transmit stereo digital audio signals between consumer audio equipment.

SOPC Builder Component: a module in SOPC Builder that contains the hardware and software necessary to access the corresponding hardware peripheral.

SOPC Builder: software that provides a GUI-based system builder and related build tools for the creation of FPGA-based subsystems, with or without a processor.

Surface Mount Technology (SMT): is the technique or manufacturing process used to mount electronic products when the Surface-Mount (SMT) are directly placed onto the surface of the printed circuit boards (PCBs).

Synchronization: is the process to settle to a fixed value during a time a metastable-prone (See Metastability) signal in order to make the signal appear synchronously to the incoming device.

System interconnect fabric (SIF): an interface which the NiosII processor uses to communicate to on- and off-chip peripherals.

System-on-a-Chip (SoC): generally refers to a component that integrates a lot of different functionalities typically found in several components in the same package.

Transition Minimized Differential Signaling (TMDS): is a technology used on HDMI interfaces that is based on a Low-Voltage Differential Signaling (LVDS) system, which transmits information as the difference between the voltages on a couple of wires, allowing the transmission of serial data at very high-speeds. The transmitter incorporates an advanced encoder to reduce electromagnetic interference (EMI) to allow faster transfers.

Video Field: is defined as the period from the leading edge of a vertical sync (Vsync) pulse to the same edge of the next Vsync pulse.

Video Format: the group of information required by the sink to properly display an image. Usually is defined only by a video timing (See Video Timing) and a picture aspect ratio (See Picture Aspect Ratio), but a complete definition includes a color space, a quantization range and a color component depth.

Video Frame: is one of the many still images which compose a video. In interlaced timings a video frame contains two video fields (See Video Field) while in progressive timings the two terms are synonymous.

Video Identification Code (VIC): an integer value used to identify a particular Video Format listed in CEA 861-E specification.

Video Timing: the waveform associated with the video format (See Video Format)

Bibliography

[1] Scheiber, Stephen F., *Building a Successful Board-Test Strategy. Second Edition*. Oxford: Newnes Butterworth-Heinemann. 2001

[2] Teradyne, *Designing Test Strategies for Modern PCB Assembly, 2002*, [online] Available at: http://www.teradyne.com/atd/resource/recording/testStrategy/testStrategies.pdf

[3] University of Bolton, *Online postgraduate courses for the electronics industry*, October 2007, [online] Available at: http://www.ami.ac.uk/courses/topics/ [Accessed 24 May 2011]

[4] University of Oslo, *Courses FYS4260/FYS9260 Microsystems and Electronic Packaging & Interconnection Technologies*, 2011, [online] Available at: http://tid.uio.no/kurs/fys317/ [Accessed 24 May 2011]

[5] Agilent Technologies, *Implementing Bead Probe Technology for In-Circuit Test: A Case Study*, 2007, [online] Available at: http://cp.literature.agilent.com/litweb/pdf/5989-8134EN.pdf

[6] Xilinx, *Comparing and Contrasting FPGA and Microprocessor System Design and Development*, July 2004, [online] Available at: http://www.xilinx.com/support/documentation/white_papers/wp213.pdf

[7] Altera Corporation, *Embedded Design Handbook (ver 2.7)*, March 2010. [online] Available at: http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf

[8] Altera Corporation, *Quartus II Handbook. Volume4: SOPC Builder (ver 10.0)*, July 2010. [online] Available at: http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf

[9] Altera Corporation, *Avalon Interface Specification (ver1.3)*, August 2010. [online] Available at: http://www.altera.com/literature/manual/mnl_avalon_spec.pdf

[10] Altera Corporation, *Nios II Hardware Development Tutorial (ver 3.0)*, December 2009. [online] Available at: http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf

[11] Altera Corporation, *Nios II System Architect Design Tutorial (ver 1.0)*, June 2009. [online] Available at: http://www.altera.com/literature/tt/tt_nios2_system_architect.pdf

[12] OpenCores, *Wishbone System On-Chip (SOC) Interconnection Architecture for Portable IP Cores Rev.B.3*, September 2002, [online] Available at: http://cdn.opencores.org/downloads/wbspec_b3.pdf

[13] On Semiconductor, *NBC12429: 3.3 V / 5 V Programmable PLL Synthesized Clock Generator (25 to 400 MHz) Rev. 13*, January 2009, [online] Available at: http://www.onsemi.com/pub_link/Collateral/NBC12429-D.PDF

[14] Herveille, Richard, *I2C-Master Core Specification Rev. 0.9, July 2003*, [online] Available at: http://opencores.net/svnget,i2c?file=%2Ftrunk%2F%2Fdoc%2Fi2c_specs.pdf

[15] Phillips Semiconductors, *The I2C-bus specification version 2.1*, January 2000, [online] Available at: http://www.nxp.com/documents/other/39340011.pdf

[16] LaCie, *Color Management White Paper 3: Color Spaces & Color Translation*. 2007. http://www.lacie.com/download/whitepaper/wp_colormanagement_3_en.pdf

[17] Jack, Keith, *Video Demystified. A Handbook for the Digital Engineer 4th Ed*. Oxford: Newnes Elsevier. 2005.

[18] Katz, David J. and Gentile, Rick, *Embedded Media Processing*, Oxford: Newnes Elsevier. 2006.

[19] Consumer Electronics Association (CEA), *CEA-861-E: A DTV Profile for Uncompressed High Speed Digital Interfaces*, March 2008

[20] HDMI Licensing LLC, *High-Definition Multimedia Interface Specification Version 1.4*, June 2009

[21] AudioQuest, *HDMI Demystified*, October 2007. [online] Available at: http://www.audioquest.com/resource_tools/downloads/whitepapers/HDMI_Demystified-v2-72d.pdf

[22] NXP, *TDA 9989 Datasheet*, June 2009. [online] Available at: http://www.nxp.com/documents/data_sheet/TDA9989.pdf

[23] Analog Devices, *ADV7511 Low-Power HDMI 1.4 Transmitter with Audio Return Channel. Preliminary Technical Datasheet Rev. PrA*, November 2009

[24] Analog Devices, *ADV7511 Low-Power HDMI 1.4 Transmitter with Audio Return Channel. Hardware User's Guide Rev. PrB*, November 2009

[25] Analog Devices, *ADV7511 Low-Power HDMI 1.4 Transmitter with Audio Return Channel. Programming Guide Rev. A*, March 2010

[26] Analog Devices, *AD9889B High Performance HDMI/DVI Transmitter Datasheet*, July 2007

[27] Analog Devices, *AD9889B/AD9389B Programming Guide Rev PrA*, July 2007

[28] Analog Devices, *SDUG-AD9889A Software Driver User Guide Rev. 0*, 2006

[29] Analog Devices, *AN-810 Application Note. EDID and HDCP Controller User's Guide for the AD9889*, 2006

[30] Quantum Data, *Implementing EDID That Works*, September 2007. [online] Available at: http://www.quantumdata.com/pdf/edid.pps

[31] Benjamin Possolo, *HDCP over HDMI: An introduction to HDCP and common attacks against it*, Seminar ATDS (WT05/06), Universität Stuttgart, January 15, 2006.

[32] Silicon Image, *HDCP: what it is and how to use it*, April 2002. [online] Available at: http://www.siliconimage.com/docs/Press_PDFs/223300.pdf

[33] Digital Content Protection LLC, *High-Bandwidth Digital Content Protection System Rev. 1.1*, June 2003

[34] Extron Electronics, *HDCP – A Technical Overview Whitepaper Rev. 1.0*, November 2009. [online] Available at: http://www.extron.com/download/files/whitepaper/hdcp_wp.pdf

[35] Doulos, *A counter for fast events, using a Flancter*. 2009, [online] Available at: http://www.doulos.com/knowhow/fpga/fastcounter/ [Accessed 24 May 2011]

[36] ASIC World, *Interfacing Two Clock Domains*, March 2011, [online] Available at: http://www.asic-world.com/tidbits/clock_domain.html [Accessed 24 May 2011]

[37] Cadence, *Clock Domain Crossing*, 2004, [online] Available at: http://w2.cadence.com/whitepapers/cdc_wp.pdf

[38] Texas Instruments, *Metastability Performance of Clocked FIFOs. First-In, First-Out Technology*. 1996. [online] Available at: http://focus.ti.com/lit/an/scza004a/scza004a.pdf

[39] Drange, Geir, *I²S Interface Specification Rev. 1.0*, January 2005, [online] Available at: http://opencores.net/project,i2s_interface,overview

[40] Phillips Semiconductors, *I²S bus specification*, June 1996, [online] Available at: http://www.nxp.com/acrobat_download2/various/I2SBUS.pdf

[41] Texas Instruments, *PCM175x: 24-Bit 192 kHz Sampling Delta-Sigma Audio DAC (Rev. C)*, February 2009, [online] Available at: http://focus.ti.com/lit/ds/symlink/pcm1754.pdf

[42] Texas Instruments, *TAS3308 Digital Audio Processor With Analog Interface Preview Datashet*, November 2007.

[43] Wikipedia: The Free Encyclopedia, *Pulse-Code Modulation*, June 2011, [online] Available at: http://en.wikipedia.org/wiki/Pulse-code_modulation

[44] Schexnayder, Brandon M., *Reshaping Digital Audio: DSD encoding as a viable alternative to PCM*, Fall 2004, [online] Available at: http://sonido.uchile.cl/articulos/Reshaping_Digital_%20Audio.pdf

[45] Analog Devices, *A Technical Tutorial on Digital Signal Synthesis*, 1999. [online] Available at: http://www.ieee.li/pdf/essay/dds.pdf

[46] Analog Devices, *Ask The Application Engineer 33: All About Direct Digital Synthesis*, August 2004, [online] Available at: http://www.analog.com/library/analogdialogue/archives/38-08/dds.pdf

[47] Altera Corporation, *Quartus II Help: Memory Initialization File (.mif),* 2007. [online] Available at: http://quartushelp.altera.com/9.1/mergedProjects/reference/glossary/def_mif.htm

[48] Altera Corporation, *Guidelines for Developing a NiosII HAL Device Driver v3.0,* January 2010. [online] Available at: http://www.altera.com/literature/an/an459.pdf

[49] Labrosse, Jean J., *MicroC/OS-II The Real-Time Kernel 2nd Edition,* San Francisco: CMP Books, 2002.

[50] Altera Corporation, *Nios II Software Developer's Handbook (ver 9.1.0)*, November 2009. [online] Available at: http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf

[51] Altera Corporation, *Using Lightweight IP with the NiosII Processor Tutorial*, December 2004. [online] Available at: http://www.altera.co.jp/literature/tt/tt_nios2_lwip_tutorial.pdf

[52] Video Electronics Standards Association VESA, *VESA Enchanced Extended Display Identification Data (EDID) Implementation Guide ,* June 2001. [online] Available at: http://www.vesa.org/vesa-standards/standards-summaries

[53] Makofske. D, Donahoo, M.J., Calvert, K.L., *TCP/IP Sockets in C#: Practical Guide for Programmers*, San Francisco: Newnes Elsevier. 2004

[54] Kurniawan, Budi, *Using .NET Sockets*, October 2002. [online] Available at: http://ondotnet.com/pub/a/dotnet/2002/10/21/sockets.htm

[55] Dhar, Ashish, *Socket Programming in C# - Part II*, July 2003. [online] Available at: http://www.devarticles.com/c/a/C-Sharp/Socket-Programming-in-C-sharp-Part-II

[56] National Instruments, *Process Model Theory*, January 2010. [online] Available at: http://zone.ni.com/devzone/cda/tut/p/id/3819

[57] National Instruments, *About the TestStand Process Models*, September 2006. [online] Available at: http://zone.ni.com/devzone/cda/tut/p/id/2694

[58] National Instruments, *TestStand Style Guide: A Guide to Developing Effective, Maintainable TestStand Sequences*, August 2007. [online] Available at: http://zone.ni.com/devzone/cda/tut/p/id/4267