Platform Independent Debug Port Controller Architecture with Security Protection for Multi-Processor System-on-Chip ICs

Dimitry Akselrod[a,b], Asaf Ashkenazi[a] and Yossi Amon[a]
[a]Freescale Semiconductor Israel, Ltd., Herzlia, Israel, 32000
[b]ECE Dept., McMaster University, Hamilton, ON, Canada, L8S 4K1
Email: adimitry@grads.ece.mcmaster.ca

Abstract

A Debug Port Controller (DPC) architecture, designed for re-use in multiple System-on-Chip (SoC) Integrated Circuits (ICs) is presented. The DPC incorporates security protection against unauthorized access along with advanced debugging features such as long chain debugging, universal BIST engines control, and generic serial interfaces. An implemented security architecture of DPC is presented together with an overall IC security scheme. DPC is the most important part of this IC security scheme. The suggested architecture demonstrates extensive use of the debug process, and re-use of the DPC in multiple SoC ICs without the need of adopting its design for a specific SoC. The implementation of the DPC for IEEE1149.1 standard is presented and the hardware realization of the proposed architecture is described in detail. The DPC that incorporates the proposed architecture has been designed in a 90 nm CMOS process as an integral part of several SoC ICs.

1. Introduction

Efficient test methodologies play one of the major roles in modern System-on-Chip (SoC) designs. The constantly increasing complexity of integrated circuits has set high demands for effective test procedures. The test cost per transistor has been decreasing at a much slower rate than the manufacturing one and currently occupies a significant portion of the total cost of IC [1]. IEEE 1149.1 standard [2], which defines a serial interface to access test-dedicated logic embedded in IC, has been widely used for implementing a wide variety of debug and test functions at the wafer, packaged chip, and the board level. The majority of test and debug functions of a modern IC, including boundary scan testing, Built-In Self Test (BIST) of embedded memory modules and peripherals (and more), can be accessed through the standard debug port. Designing a Debug Port Controller (DPC) which could be either partially or completely re-used for another IC is especially desirable for SoC's [3], [4].

This paper presents a generic architecture of a DPC which can be used in a series of SoC ICs without any modifications being done to the DPC module design. The adjustment to the specific IC is done by chip-level system integration of the DPC as well as by DPC registers programming.

Special attention has been paid to the thorough security protection of the IC. As known, debug port manipulation is one of the most common ways of executing unauthorized program code, acquiring control over secure applications, and running code in privileged modes.

Section II presents the proposed architecture. Section III describes the methods of security protection used in the proposed DPC. Section IV discusses the implementation and testing details of the SoC IC. Section V concludes the paper.

2. Proposed debug port architecture

SoC IC may incorporate several microprocessors, microcontrollers, or CPUs, when as a rule each one will include its own DPC. One of the design restrictions of SoCs is that debug controllers of the integrated CPUs (or peripherals) must be taken "as-is", without modifications.

It is the SoC DPC (referred to in this paper as "the DPC") which plays a major role in controlling the overall debug process, and serves as a master debug controller. In the modern design process, companies concurrently design a series of SoC ICs all having many design aspects in common. In such a case, designing a dedicated DPC for each specific IC design is expensive and inefficient. The proposed SoC DPC has been designed so that it can be placed in a wide variety of architecturally similar ICs without any change in its design. The DPC has already been incorporated into at least five different ICs. Each of

3-9810801-0-6/DATE06 © 2006 EDAA

these ICs bear a certain similarity, but includes a different set of peripherals and microprocessors.

The majority of the features and architectural approaches described in this paper do not necessary have to be implemented using a specific standard. Nevertheless, the DPC complies with the IEEE1149.1 standard. We will therefore refer to this standard when mentioning specific details of implementation. Fig. 1 shows the connectivity scheme of the DPC with existing on-chip CPUs. All of the mentioned CPUs include a debug controller as an integral part of their design. All these debug controllers are connected serially including the DPC itself. Such connectivity is supported by the majority of debug tools currently present on the market. Some of these debug controllers may not include the standard IEEE1149.1 Test Access Port (TAP) controller and thus are unable to determine the correct TAP state for their operation. Such debug controllers need TAP states information (such as Shift-Dr, Update-Dr [2], etc) to be decoded and supplied to them. In order to provide for states interpretation for such debug controllers, the DPC monitors all TMS [2] control signals intended for each of the debug controllers in the IC. Thus, the TAP states information can be supplied

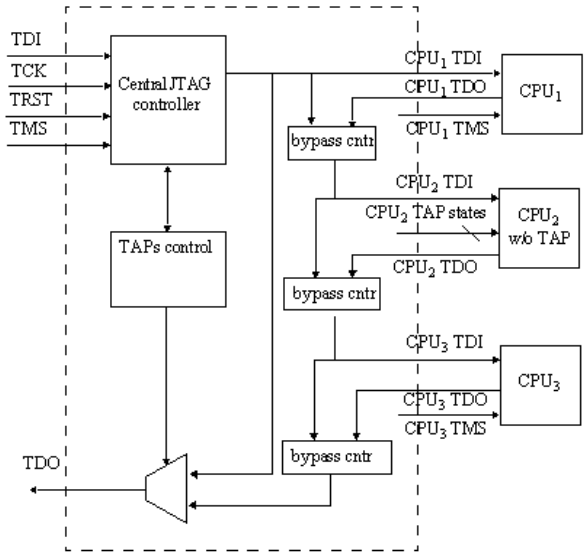

Fig. 1. DPC Connectivity Block Diagram.

for those controllers that do not have a TAP controller inside. These controls are separate for all such controllers even though for all the active controllers, TAP states and their transitions are identical. For different controllers in certain situations, it is better to supply controls different from the common TMS control sequence (this is mainly because of security reasons).

2.1 Serial Configuration and Standalone Configuration

As can be seen from Fig. 1, two basic configurations are provided - so called *Standalone Configuration* of DPC and *Serial Configuration*. *Standalone Configuration* means that only the DPC is connected between IC′s debug data ports TDI and TDO [2]. In *Serial Configuration* all the debug controllers on the chip are connected in one chain. The latter configuration is supported by the majority of commercial debug SW tools on the market and is convenient for on-board testing and debugging of the IC and the overall system; whereas the first one is convenient when performing post-silicon testing of IC wafers, and can substantially shorten testing time. For example, the majority of on-chip memory modules are tested using on-chip Memory Built-In-Self-Test (BIST) units (which are controlled by a DPC). Using only one DPC for the chip allows the tests to be carried out in the most efficient manner, and improves the quality of the test itself. *Standalone Configuration* is especially effective for post-silicon testing because the data and instruction chains are the shortest in this mode as they include only DPC.

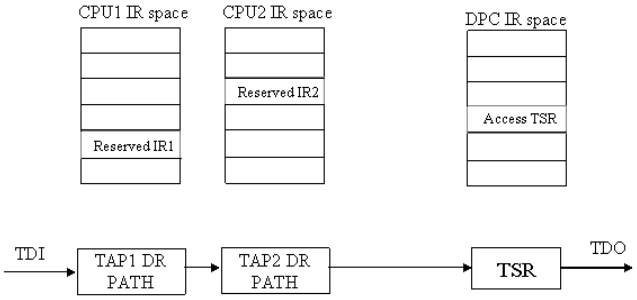

Fig. 2. Mapping "Access TSR" instructions in CPU′s TAPs reserved IP spaces.

Fig. 2 shows a typical debug chain consisting of several debug port controllers together with the dedicated SoC DPC. At least one Instruction Register (IR) must include an *Access TAP Select Register* (*Access TSR*) instruction. *Access TSR* instructions are included in the SoC DPC IR space. In addition to the SoC DPC, this instruction can be added to other DPCs′ IR spaces, in place of available reserved IR spaces. It is the additional logic, present in SoC DPC, which tracks the states of all DPCs′ IR registers and notifies SoC DPC in case any of on-chip CPUs has a *Access TSR* instruction in its DPC′s IR. The latter implementation allows the original DPC to be integrated as is. TSR register is accessed via a Data Register (DR) path. If more than one *Access TSR* instruction is given, only one will be chosen based on a pre-defined priority.

Each bit in the TSR controls the bypass of a corresponding DPC in the IC. Post reset configuration of the DPCs may vary, set by input ports sampled at reset, or hard coded in the SoC.

When a certain DPC is bypassed, it is replaced by a dummy IR register of the same length as the original one in the IR path; and by a one-bit DR register in the DR path as shown in Fig. 3. In this situation, the debug software (in the IR path) will still observe the same length IR register, so no special software support is needed for this feature. One of the possible applications of bypassing a DPC, besides shortening the overall DR path length, can be increasing the debug clock frequency. Some of the peripherals in the chain may allow for much higher debug clock frequency rates than the others. In this situation, in order to switch to a higher frequency, we would need to exclude from the debug chain those peripherals that cannot support such frequencies.

2.2 Accessing SoC DPC′s registers

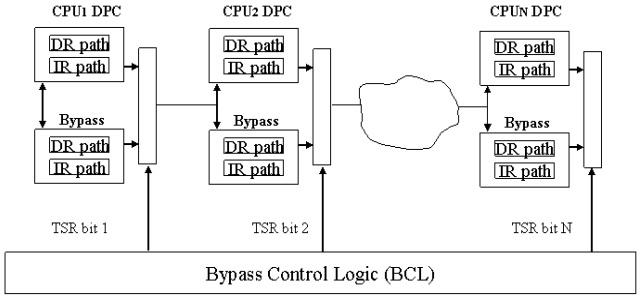

Fig. 3. Serial Configuration with Bypass Control Logic.

SoC DPC IR has 5 bits, which gives total of 32 different instructions coded with these bits. One of possible instructions for example could be *Access TSR*. Additional instructions could include standard IEEE1149.1 instructions, as well as others. Taking into account a number of features implemented in the SoC DPC, a total of only 32 IR instructions, addressing up to 32 different registers might lead to an IR instruction shortage. One of the possible solutions could be increase the length of the DPC IR, but this solution would increase the time needed to program the IR register by about 20instruction size added). The solution we have used is to have a number of additional registers mapped under the same IR instruction. For this purpose one of the IR instructions is used to access the additional 32 registers which we will refer to as *ExtraDebug registers*. The instruction is called Access ExtraDebug Registers (AEDR). Naturally there can be foreseen multiple AEDR instructions in the IR address space ($AEDR_1$,$AEDR_2$, etc.), each one controlling its own different set of *ExtraDebug registers*. The AEDR instruction technically gives an access to an additional register, called the *ExtraDebug Shift Register* (EDSR) which consists of 38 bits comprising a 32 bit data field, a 5 bit address field and read/write bit.

2.3 ExtraDebug registers write access

The write access to any of *ExtraDebug Registers* takes place when the TAP controller of the DPC enters the *Update-DR* state. At that stage the lowest 32 bits of data of the EDSR are written to the *ExtraDebug registers* referenced by the 5 bit address field of the EDSR, provided that the read/write bit of EDSR is negated.

2.4 ExtraDebug registers read access

The read access is divided into two cycles. In the first cycle the data field of EDSR is ignored. Provided that read/write bit of EDSR is asserted, the data from the corresponding *ExtraDebug register* will be fetched by the second read access cycle. The read access will be accomplished on the next cycle in the Capture-DR state, and the data will be shifted out during the Shift-DR state. In the second path for a read access, simultaneous write access is not supported. The number of Shift-Dr states during the second cycle can vary depending on the width of the accessed register

2.5 Generic serial channels

The DPC provides a number of generic serial access channels which allow access to SoC modules through a serial (*TDI/TDO*) interface. In order to access an arbitrary serial channel, *Serial Access Select Register* (*SASR*), one of the *ExtraDebug registers* should contain the address of the desired serial channel to be used. Naturally, only one serial channel can be chosen and accessed at a given time. Only after this access can a *Serial Access* instruction be entered into the IR register. Once such an instruction is entered into the IR register, any entry to *Shift-DR* state will generate one cycle of the control signals for the chosen serial channel to be activated and the corresponding serial channel output multiplexed to TDO output (as all the serial channel inputs are already connected to the common TDI input). Looping in *Shift-DR* state will produce a required number of serial access cycles.

A number of serial channels are configured to be secured, i.e. the secured serial channels will be available only in corresponding security modes.

In the current implementation of the SoC DPC, several

serial channels are split. Each such channel is split to a number of different serial channels intended for BIST test related information access. The specific channel in this case is chosen taking into account additional information residing in *BIST Configuration Registers* (*BCR*).

2.6 Universal BIST Configuration Registers

One of the problems arising during DPC design is the uncertainty in designing BIST Configuration Registers (BCR). Custom designing BCR registers so that they would rigidly correspond to the specific BIST engines in SoC has obviously numerous disadvantages. For example, in one of the ICs that SoC DPC has been utilized in, there are several different BIST engines each one having its own design, architecture, and interface. The solution is thus having several generic BCR registers each having the same structure: General Purpose (GP) bits which could be accommodated for connecting to and controlling any possible interfaces in BISTs, N-bit *SelectBus* bus, *Invoke* signal, and *ReleaseEnable* signal. N defines a maximal number of identical BIST engines controlled by one BCR register and it could be different for different BCR registers. From our experience any BIST engine having a reasonable yet custom interface could be controlled by such BCR interface. The custom-defined bus and two other controls are involved in:
- Forming *Select* control for all the BIST engines from the same group.
- Forming individual *Release* signals for each one of the BIST engines from the same group. *Release* signal is used to release BIST from several possible modes when it is stopped, such as *Retention Test Mode* or *Debug Mode*. The individual *Release* signal for a BIST engine is generated whenever *Serial Access* instruction resides in IR register, *ReleaseEnable* signal is asserted, *SASR* register contains corresponding serial channel number corresponding to the specific BIST engines group, and TAP controller is in *Update-DR* stage.

The proposed solution allows simple and flexible interfacing of a wide range of BIST engines inside an SoC.

2.7 Debug Control and Status Support

DPC controls some of the on-chip processors′ debug mode features. This includes gating and routing of debug mode entry requests coming from various on-chip and external sources, SW induced debug mode entry for a number of supported microprocessors, as well as providing status information regarding debug state of on-chip processors.

3. Security protection

Debug port manipulation is one of the known ways of executing unauthorized program code, gaining control over secure applications, and running code in privileged modes. Debug ports (e.g. IEEE 1149.1), provide a debug access to a number of hardware resources, among them the system processor and the system bus. This could lead to losing program control as well as providing unwanted visibility into system peripherals and memory units. Debug port access provides an intruder with all the means required to break the system′s security mechanisms and gain control over the operating system. Unauthorized debug port usage should be prohibited in order to properly secure the system. However, the debug port must be available for platform development, manufacturing tests and troubleshooting, as well as for software debugging by authorized entities.

A security architecture aimed to prevent debug port manipulation while allowing access for manufacturing tests and software debugging has been developed thus forming an access regulator to IC debug resources [9].

3.1 Security modes

This regulator provides several different protection levels represented by four corresponding modes. The required mode is selected by electrical fuses' (e-fuses) configuration and is programmed according to the product state (e.g. manufacturer level quality testing, OEM mode, end user mode, etc.), and the specific customer security requirements. The four debug port security modes are defined as follows:

Mode 1: No Debug (Maximum Security)
This mode provides the highest security level. When selected, all debug port features are disabled, including (but not limited to):

- Processor run time control - processor execution thread stop and single-step operation.

- Memory access - read/write access to processor registers, peripheral memory space or on-chip memory modules. Memory accesses refers also to so-called on-the-fly debug access mode in which the memory access is performed while the processor is running and is not in the conventional debug mode.

- Advanced modes of *Memory Built In Self Test* (BIST) operation, specifically so called *BIST debug mode* which allows advanced control of BIST operation which could result in exposure of protected memory regions.

- Boundary Scan, which is as defined by the IEEE 1149.1 standard, is an integrated method for testing interconnects on printed circuit boards that is implemented at the IC level.

A Scan test is a powerful test technique, but at the same time it could also be used as a powerful attack tool. Scan chains can be used to recover on-chip secret keys, which are not memory mapped, but could definitely be accesses during scan operation. Yang, et. al. [7] show how a scan channel can be used as a side channel to recover secret keys from a hardware implementation of Data Encryption Standard (DES). A Scan will be available in the No Debug mode but the designed debug controller is provided with a mechanism that detects an entry into the scan test modes. This mechanism is not a part of the scan chain therefore cannot be affected by a scan side based channel attacks. Whenever scan mode entry is detected, the debug controller alerts the on-chip secure modules, which in response reset all of their internal flip-flops. This solution allows for testing of the secure memory flip-flops while eliminating scan based side channel attacks. Control operation which does not jeopardize overall IC security will be allowed, including various PLL bypass controls, status bits external visibility, etc. These features do not reduce the security level of the product, and they allow performing important tests and board connectivity checks.

Mode 2: Secure Debug (High security)
This mode limits the debug port access by using an authentication protocol. Any access to the debug port is verified and authenticated. Only authorized debug tools, possessing the correct secret key, are able to take advantage of the debug port features. The secret key can be unique for each individual chip, or for a small group of chips.

Mode 3: Debug port Enabled (Low Security)
All debug port features are enabled. Although no access authentication is required, a debug alarm will be activated when a debug port access is detected

Mode 4: Development (No Security)
All debug port features are enabled, and no alarm is activated when debug port access is detected. In order to allow development of security applications that take advantage of the systems security modules, it is required to allow unlimited debug access to all the systems resources, including security modules. This mode, of course, eliminates any system protection and therefore is configured only in special samples used in development kits and not in merchant market standard devices.

3.2 Fuse configuration

The debug security modes are configured using e-fuses which can be burned after packaging merely by applying electrical signals. An e-fuse can be burned at any time and can give the OEM vendor the flexibility of deciding the security level of its products. The fuse burning is an irreversible process; once the fuse has been burned it is impossible to return the fuse back to the un-burned state.

By burning a specific fuse to the ”Debug Port Bypass” mode, it is possible to bypass the ”Secure Debug” mode, thus changing a device’s state from ”Secure Debug” to ”Debug port Enabled” mode. In all other security modes this fuse will have no effect. This feature allows the user to permanently open up the debug port access for testing, without the need to undergo an authentication process after each reset. In some cases the device under test is taken from a customer which might have reported certain issues related to it, and is to be returned to the customer. In other cases, a second fuse, ”Cancel Debug Port Bypass”, will be burned. The IC will then be reverted back to ”Secure Debug” mode. Once ”Cancel Debug Port Bypass” is burned, there is no way to activate Secure Debug Port Bypass again.

3.3 Authentication protocol mechanism in Secure Debug mode

The authentication protocol mechanism uses a ”Challenge-Response pair”. The ”challenge” is a number that identifies the specific SoC. The ”response” is a number that must be sent in response to receiving the ”challenge”. When a query is sent to the SoC by an outside source, it receives the ”challenge”, and must respond with the correct ”response”. If the response is incorrect, the SoC DPC will not allow access to DPC debug features. Fig. 4 shows the process of challenge-response exchange, when first the challenge code is requested and supplied, and then the correct user response is fed inside the DPC negating the *Secure Block* signal allowing access to security sensitive parts of the IC.

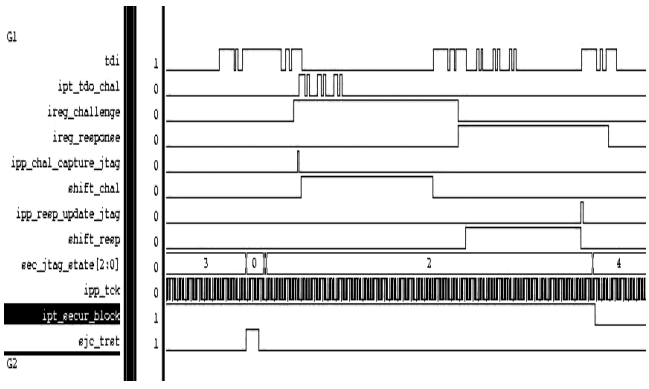

Fig. 4. Challenge-response exchange process.

There are two modes that the DPC supports for generating the Challenge - Response pair. Fixed challenge-response pair: each part has its individual challenge - response pair which is determined at manufacturing time, and will not change later on. The DPC will compare the users response to the expected response. Random challenge-response pair: in this mode, a random challenge will be generated by a separate module, and the response, which is a cryptographic manipulation of the random challenge, will be checked inside the DPC as well. The DPC can be configured to both modes.

3.4 Security State machine intervention prevention

The DPC security scheme will prevent attempts to tamper with its Finite-State-Machines (FSM) which could result in setting them to unprotected states skipping the proper authentication procedures. It is always assumed that Power-on-Reset (PoR) is asserted thus setting FSMs to secured initial states. This assumption may not hold true as PoR detection is usually placed outside the IC, thus an intruder may switch off and on the IC power multiple times thus accidentally causing one of DPC′s state machines memory elements to reach a combination which corresponds to an unprotected state. For example, if an FSM has 4 different states, they are defined by using only two bits. It means that if we could cancel PoR assertion at power-up, several tries could lead to these two bits receiving a value which corresponds to an unprotected state. In order to prevent it, we have introduced a register, all the bits of which turn to a specific pattern only after the PoR reset. None of unprotected states of DPC′s state machines can be reached while this register value is not equal to the pre-defined value. As this register′s length can be 64 bits or more, the probability of it reaching this value accidentally is minor.

4. Implementation details

The SoC DPC described in this paper has been integrated in a number of IC′s, among them Freescale′s i.MX31 application processor. Fig. 5 shows the application processor′s development board that connects to a commercial JTAG-based debug interface.

The application processor is based on the $ARM1136JF\text{-}S^{TM}$ core, starting at 532 MHz, with a vector floating point co-processor and L2 cache. It is fabricated using 90nm CMOS technology and is designed for use in wireless devices running computationally-intensive multimedia applications such as digital video broadcast and video conferencing.

5. Conclusion

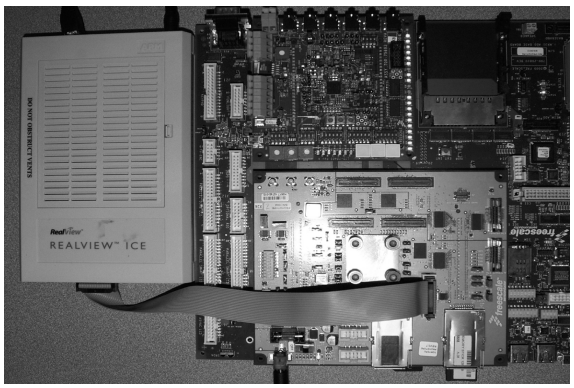

Fig. 5. Application processor′s development board.

A Debug Port Controller (DPC) architecture designed for re-use in multiple SoC ICs has been presented. The suggested architecture shows facilitation of the debug process and provides a flexible means for incorporating the DPC in different SoC ICs without modifying its design. Detailed architectural description of the proposed DPC has been shown and the fabricated IC incorporated has been presented. Implemented security protection scheme which is the integral part of the DPC has been presented and analyzed.

References

[1] T. Rahal-Arabi, G. Taylor, *A JTAG Based AC Leakage Self Test*, VLSI Circuits Digest of Technical Papers Symp., 2001, pp. 205-206

[2] *Standard Test Access Port and Boundary-Scan Architecture*, IEEE standard 1149.1-2001, 2001

[3] D. Y. Jung, S. H. Kwak, M. K. Lee, *Reusable embedded debugger for 32 bit RISC processor using the JTAG boundary scan architecture*, Proc. 2002 IEEE Asia-Pacific Conference on ASIC, pp.209 - 212, Aug. 2002

[4] Ingeol Chun; Chaedeok Lim, *ES-debugger: the flexible embedded system debugger based on JTAG technology*, Intl. Conf. on Advanced Communication Technology (ICACT 2005), Volume 2, pp. 900 - 903, Feb. 2005

[5] Y. Amon, D. Akselrod, E. Segev, *Integrated Circuit and a Method for Testing a Multi-Tap Integrated Circuit*, an International patent application, PCT/EP2004/014805, filed on Nov. 22, 2004

[6] A. L. Crouch, *Design-for-Test for Digital ICs and Embedded Core Systems*, Prentice Hall PTR, pp. 93-174, 1999.

[7] B. Yang, K. Wu, R. Karri, *Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard*, IEEE International Test Conference (ITC), October 2004.

[8] W. Moyer and M. Fitzimmons, *Integrated Circuit Security and Method therefore*, Patent application No. 10/100,462, Pub. # US 2003/0177373 A1, Publ. Date Sep. 18, 2003, Motorola Inc

[9] D. Akselrod, Y. Amon, A. Ashkenazi, *Integrated Circuit and a Method for Secure Testing*, an International patent application, PCT/EP2004/014804, filed on Nov. 22, 2004

[10] A. Ashkenazi, *Securing Smartphones from the Inside Out*, Design Seminars 2005 Proceedings, Embedded Systems Conference San Francisco 2005, 3G Cellular Design Seminar, session 3GC-702.