Design of a 1-chip IBM-3270 Protocol Handler

L. Spaanenburg *)

Twente University
Department of Electrical Engineering
P.O. Box 217
7500 AE Enschede The Netherlands

Abstract. The single-chip design of a 20MHz IBM-3270 coax protocol handler in a conventional 3 μ m CMOS process-technology is discussed. The harmonious combination of CMOS circuit tricks and high-level design disciplines allows the 50k transistor design to be compiled and optimized into a 35 mm**2 chip in 4 manweeks. The design methodology stresses the application of high-level silicon constructs and built-in testability.

Keywords. ASIC, logic-enhanced memory, VLSI design, microarchitecture, built-in testability.

0. Introduction

The IBM-3270 protocol relates to the exchange of control & data messages between an IBM computer and a collection of intelligent peripherals over a serial coaxline (see Fig.1). The protocol is constantly being adapted, as more and more intelligent peripherals become available. Non-IBM equipment can therefore only be connected through a protocol handler, which can easily be altered. In the past this has been realised by means of a general-purpose microprocessor in combination with either a gate-array or PLD's. Recently the DP8344 offered a programmable 1-chip solution based on an extended general-purpose microprocessor architecture. This paper discusses an alternative to optimize the throughput.

Fig.1 Typical configuration using the IBM-3270 protocol

It is generally acknowledged, that full-custom design provides for an optimal design, yet at the expense of design time. Standard-cell could provide a compromise, but often lacks sufficient sophistication to benefit fully from the design structure. This can be improved by introducing standard structures above the gate level. In the paper the *function cell* will be introduced. As shown, this allows to retain the design structure, while reaching a speed of design similar to full-custom.

In the first paragraph the system architecture will be outlined. Then the function cell is introduced and it is shown how a system specification in the C-programming language corresponds to a collection of function cells. Ensuing it will be discussed how testability can be accommodated for. In conclusion the overall design process will be discussed.

Fig.2 Protocol handler based on the LSID904

1. System architecture

The IBM-3270 protocol handler interfaces between a coaxline network and the custom application. The application needs a screen memory, where the symbols to be printed/imaged are written together with attribute and font info. Part of this information is pre-stored such as character fonts; part of it is constantly changing. Furthermore the application is microprocessor based to serve the actual peripheral. The connection to the LSID904 screen memory must be claimed from and granted by the LSID904. Response time depends on the priority of the command currently executed by the protocol handler.

* Current address:
Institut für Mikroelektronik Stuttgart,
Allmandring 30a, 7000 Stuttgart 80 (B.R.D.)

Figure 2 shows, that the LSID904 acts as an intelligent arbiter between several datastreams. Additionally control messages are exchanged between the parts of the system. Control messages are prioritized.

The functionality of the protocol handler can therefore be listed as:

- traffic arbiter & switch
- protocol conversion
- memory management

From this, a functional diagram can be derived as shown in figure 3. On the right-hand side the coax handler is located, which provides the physical and datalink layer for

reading/writing IBM-format coax messages. On the left-hand side the physical layer for management of the local memory is located. It provides programmable schemes for address translation. A next item is the programmable and prioritized control store, which translates control messages in IBM format into a set of nano instructions according to the functionality of the protocol handling innerpart. Furthermore it provides a first level of datastream arbitration. Internally hardware is provided for virtual memory addressing, base & extended base data handling, interrupt & watchdog handling, poll & status bookkeeping and arbitrary width data manipulation.

Fig.3 Functional diagram of the LSID904

The coaxline carries 12 bits messages. The embedded 8-bits words are retrieved and sent to the sequencer for controlwords or to register locations designated by the controlstore for datawords. IBM-format controlwords are translated into one or more internal commands. For the internal transfer of information, 3 busses are present:

- a 3-wire timing bus for system timing
- an 8-wire control bus for addressing internal storage location and for specifying a internal command
- an 8-wire databus for carrying data & external memory address information.

Fig.4 Internal timing diagram
(For asynchronous message transfer,
the time periods can be arbitrarily enlarged)

The internal system timing is based on 4 cycles. First a source of data is identified. Then a destination for data is identified. Ensuing the transfer of information on the in-

ternal busses is accomplished and finally some internal bookkeeping is performed, while the next internal command is interpreted. In case the screen memory is destination the first 2 cycles are interchanged to reduce the chance on memory wait states.

2. Function cell

The archetype of the function cell is Clark's macromodule [Cl67]. In a network such modules pass activity in the order, in which they are wired. In the Register Transfer Modules [BeEg72] of Bell a distinction is made between processing modules and evoke modules. The evoke modules are cabled in the triggering order and each of them in turn activates one or more processing modules. Such concepts clearly resemble the modules in current local area networks and have received attention in the realm of VLSI where integration stepped over the limits of isochrony. A fore-runner of the function cell, as treated in this paper, appeared in [SpDu84].

A function is the hardware equivalent of the software notion "function". It is composed of (a) a timing control part, where the cell is (de)activated, (b) a function control part, where the control messages are decoded and (c) a data part, where the actual computations are performed. These three parts are similar to the header, case and body

of the software function. In figure 5 an example is depicted. Note that the function cell concept is based on the observation, that the structural denotations "controlpath" and "datapath" relate to the use of the logic. For example, the controlpart holds the command decoder as datapath. As a result the function cell operates best for systems with distributed control.

Fig.5 Composition of the "function cell"

- denotes an IO-latch
- denotes a control structure
- denotes a data structure

Another way to look at the function cell is by its floorplan (Fig.6). It is largely based on the orthogonality of the 3 layers. The timing bus is fed to the start macro, which takes part of the control information to see whether the hardwired address of the cell is identical to the address in the controlword. If this is the case, the command is fetched from the controlbus and decoded. Once the command is decoded, the I/O latches of the datapart are opened and data is transferred between function cells.

Fig.6 Template of a function cell

In the LSID904 16 function cells of 8 different types are accommodated for. Complexity varies from an address counter together with 4 boundary registers/comparators to a straightforward 8 register scratchpad. Furthermore function cells are added to the physical layers and the control sequencer to provide a uniform handling of data and control throughout the system.

The target technology is a 3 μm CMOS process with single-layer metal. From the orthogonality of data- and controlpart it follows, that the controllines will run for several millimeter in polysilicon. This leads to not acceptable transfer delays. A special cell has been developed to refresh the control signal at regular intervals (Fig.7). The driver D1 is dimensioned large enough to draw the line low. However if the line is to carry a logic '1', it will slowly start loading the line. Once this signal level passes the threshold of the feedback inverter, the other Pmost P1 will become conducting providing a very fast signal edge. Although the inertial switching delay is thus enlarged, the overall propagation delay is reduced. In our case, placing a repeater every 1.2 mm brings for the largest function cell the access time down from 178 nsec to a comfortable 38 nsec.

Fig.7 Repeater cell for long polysilicon lines

The function cell template can be characterized as a *logic-enhanced memory*. By large all the cells about as in a memory; at the crosspoints not only memory devices but also combinatorial logic is present. The logic design is therefore largely dominated by a pitch problem: as a result, assuming the size of the datapart components to be fixed, the shape and the complexity of the function control cells is bounded.

Several variations on the above template are in existence, such as:

- **backfire configuration.** Here the (primary) function control not only activates the information transfer but also a secondary function control. This latter control

structure allows for a degree of post activity, for instance for refreshing register contents.

- twin configuration. Here both the datapart and the function control part are doubled. The total then corresponds to one function cell but with two addresses. This largely circumvents the function control pitch problem.
- bi-face configuration. Here the timing, primary and secondary function control macros are doubled. This corresponds to one function cell with post-activity and two addresses.

From the template of the function cell, it is clear that a direct correspondence can be maintained between the C-language description of the macro composition, if one takes care to structure the program in the same manner as it is envisaged to assemble the macros.

3. Physical interfaces

The memory management unit largely consists of a set of offset registers, of which the content will be merged with the content of the current address counter. Only a small timing controller is required.

The coax unit is much more complicated. On the receiving side a Manchester-II coded serial datastream has to be decoded, simultaneously checking for parity, and will be parallelly distributed dependent on the content of the message. On the transmitting side a parallel word has to be serialised, parity has to be added and the result has to be shifted out serially. In the Manchester-II code data items require also mid-transitions. Hence to maintain the required accuracy in sampling, the datarate is 1/16 of the system clockrate. Especially the error handling makes the specification complicated.

In order to contain the design complexity and ease the specification of internal parallelism, the coax unit is described by means of hierarchical state diagrams. This is similar to timed Petri nets. An illustration of the use of hierarchy in state diagrams is shown in figure 8. At the root of the hierarchy lies a state diagram where the transition conditions relate directly to the state of the underlying diagrams. Each state is detailed by such lower level diagrams. This approach not only supports a step-by-step development, but also allows for parallelism as more than one state diagram may be activated from the higher level. The hierarchical specification is partitioned and the layout can be based on the assembly of parts of comparable size. This circumvents the granularity problem, that often gives a standard-cell layout an indented outlook [YoNa86]. In the coax unit, the transition conditions often comprise of many variables. This not only gives rise to a routing problem, but also to a delay penalty because of the increased

Fig. 8 Part of the hierarchical specification of the coax transmitter function

wiring length. For this reason the implementation is based on state cells [SpSm85], which separates transition logic from sequencing logic. Combined with hierarchical state diagrams, this made it possible to keep the wires relatively short.

Within the LSID904 the physical layers are directly coupled to function cells, where the datalink layer is implemented. Internally the physical layers can therefore be addressed as mere function cells. This separation of tasks, which is characteristic for the function cell concept, made it feasible to design all cells independent of one another.

4. Testability

The main testing problem during the design of the LSID904 was with respect to the function cells. The first requirement for operation of a function cell is, that it can be activated. To this purpose the function cells are addressed one after another and the result is read back over the datalines. In the next step the function control & data I/O latches are scanned. Subsequently the function control is tested by feeding the results back over the datalines. Finally the actual macro test of the datapart can be performed. The above list shows, that for a large part the function cells can be tested in a similar manner. Such a *universal test* (i.e. a test without taking the actual functionality into account) opens opportunities to add a BIST processor of acceptable complexity. Even the individual macro tests can be split into a universal test, evaluating the

- do-while construct to enable the subsequent addressing of function cells;
- conditional branches to break the test of a function

- subroutine facility to build a macro test from a collection of smaller tests.

[illegible]

5. System design

- **LSIcass** which takes a symbolic description of the coax and local commands to assemble a sorted list of commands as well as the coding tables for the chip. The input format supports macro definitions like:

wherein status is read till the pattern has occurred and the processor has handled the associated interrupt. This example also shows, that the keywords of the language are based on the IBM assembler with additionally some denotations for the communication with the processor.

tion, which can be printed or translated by conventional CAD into a waveform plot. The run documentation is very verbose, showing exactly the content of the register locations inside the addressed function cells. For example:

S-phase:

EOS3=0 EOS2=0 EOS1=0 EOS0=0

with TRIG=0 UP=1 IFS1=1 IFS0=1

EOS3=0 EOS2=0 EOS1=1 EOS0=1

Transfer:

D-bus: 0 0 0 0 0 1 0 1 0 0 0 0

A-bus: f f f f f f f f f f f f

Bookkeeping:

CCR0: 1 1 0 1 0 1 0 0

CCR1: 0 0 0 0 0 0 1 0

Command Priority 1

Transmitter: empty

Receiver: full

Next: 0 0 0 0 1 1 1 1

To limit the time to print simulation results, 4 levels of verbosity can be set.

In this design phase, use has been made of a charting technique, that is usually applied to develop large software systems: SADT [So81]. The construction of the model took 1.5 manyears. The reason for this long period was not the lack of CAD tools, but rather that by stimulating the model the understanding of the system improved, which gave rise to a number of additions and alterations. As simulating the entire sytem over and over again was very time-consuming (the documenting output of the simulator was very elaborate and could therefore lead to long print times), the design was first evaluated per function cell. Here the concept of function cell was again very usable, as their interface to the outerworld is very strict defined.

Once the model was finalised, the use of classical CAD in an integrated environment proved to be non-satisfactory, as the abutment-oriented style of design was not well supported. So a dedicated function cell composer was constructed from the primitive tools outof a toolbox [Sp87]. This not only made it feasible to construct the physical layout in a matter of weeks, but also reduced changing over to a different process technology to a mere re-design of 56 small leafcells.

6. Discussion

According to Adshead, VLSI stands for "Very Large Slips Imminent" [Ad87]. The project reported here makes no exception. The time required for pre-engineering (i.e. creating a detailed specification of a system, that can be integrated) exceeded the time set at the beginning considerably. However in looking back, one may conclude that things could have been worse, if not function cells had been used. The function cells allowed a partition, wherein the system components were independent of one another with respect to time, energy and area consumption. Furthermore function cells allowed to apply a universal test combined with independently developed specific tests. It is

anticipated, that an on-chip BIST processor might be feasible.

The LSID904 is fully programmable, yet shows all the benefits of dedicated hardware. For instance the Poll Response takes a single internal cycle, which stays well within the IBM-imposed limit of 5 μ sec. The DP8344 on the other hand, which is based on a stored-program micro-processor with dedicated on-chip peripheral circuitry, will distinctly exceed this limit. Furthermore due to the built-in locality of the function cells, the design will not only benefit directly from migrating to a process technology with smaller detail, but it is also feasible to do so without changing the implementation.

Acknowledgements

The 1-chip IBM-3270 protocol handler has been designed in collaboration with INCAA Datacom b.v. at Apeldoorn (The Netherlands) and 1ST Silicon Works at Almelo (The Netherlands). Henk Snaterse of INCAA, Jens Leenstra of Twente University and Han Speek of ISW are acknowledged for their contributions. The project is running as part of the Microelectronics Stimulation Program of the Dutch Ministry of Economic Affairs.

References

- [Ad87] Adshead, H.G., Hardware Description Languages and High-level Design, Alvey conference, Manchester (England), July 1987.
- [BeEg72] Bell, C.G. et al., The description and use of register-transfer modules (RTM's), IEEE Transact. on Computers 21, No.5 (1972) 495 - 500.
- [Cl67] Clark, W.A., Macromodular computer systems, Proceed. Spring Joint Computer Conf. 30 (Washington, 1967) 335 - 336.
- [So81] An introduction to SADT: Structured Analysis and Design Technique (SoftTech Publications, 1981).
- [Sp87] Spaanenburg, L., The open interconnect of CAD systems, IFIP workshop on Tool Integration, Paderborn (West-Germany), November 1987.
- [SpDu84] Spaanenburg, L. et al., One-chip microcomputer design based on isochrony and selftesting, Proc. IEE Electronic Design Automation Conf. 2 (Coventry, 1984) 161 - 165.
- [SpSm85] Spaanenburg, L. et al., A methodology for the fast and testable implementation of state diagram specifications, IEEE Journ. on SSC 10, No.2 (1985) 548 - 554.
- [YoNa86] Yoshimura, H. et al., 32-bit microprocessor design, in: Goto, S. (ed.), Advances in CAD for VLSI 6 (North-Holland, Amsterdam, 1986) 357 - 398.