EDA191 VLSI Computer Architecture (Datorarkitektur-Implementering)

Project assignment 2006/2007
Lars Bengtsson, Department of Computer Science and Engineering, Chalmers University of Technology

This document contains a specification of the project assignment in the course "Datorarkitektur-Implementering", EDA191, year 2006/2007, at Chalmers University of Technology, Gothenburg, Sweden. This course is electable for Electrical Engineering and Computer Engineering students in their 4:th year of study. Prerequisite knowledge is required from previous courses in Digital design and implementation, Computer organization, Computer Architecture, and digital design using VHDL.

A PC and a connected prototype board containing an XILINX FPGA circuit, memory and some other logic is available to each group. See www.ce.chalmers.se/edu/lib/fpga_lab_system/fpga.html for detailed spec of this card (can also be reached via the course home page www.ce.chalmers.se/edu/course/EDA191). A Javaprogram ("usbterm") is run on the PC for downloading of FPGA configuration and for memory inspections/modifications, and for other control of the FPGA board.

Synthesis to FPGA data is done using the PC-based CAD-tool "Synplify". Functional simulation is done using the ModelSim simulator. FPGA Place&Route is done using the Xilinx ISE tool. Synplify & ModelSim are also available on UNIX. The students may want to download the free software "WebPack" from Xilinx on their home PC. This can be found at www.xilinx.com/univ.

Table of Contents

1.0 SYMFONI Architecture Specification

This paragraph specifies the instruction set architecture and the programming model of Symfoni. It is based on this specification that each group should design their own processor. The goal is to design and implement a processor that is efficient regarding e.g. clock speed and size (no of equivalent gates) in a 300KGates Virtex FPGA circuit. The FPGA should be seen as a test vehicle for the architecture.

The groups are free to make the detailed pipeline design according to their own judgements. As long as the specification is fulfilled, and all design choices can be motivated, it is OK. As a help, fictive values of instruction statistics can be found on the course home page. Also, three simple testprograms (P0,P1 and P2) are provided to help in testing and verifying correct functional behaviour (can be downloaded from the course web page).

1.1 FPGA-Virtex prototype system

A prototype system containing an XILINX 300KGates FPGA “Virtex” circuit (XCV300), memory and some other logic is available to the groups to test the CPU. The system consists of three connected cards. A monitor processor card (using a 8051 microprocessor) is used for FPGA configuration downloading from PC, memory inspection/modification, etc.

See www.ce.chalmers.se/edu/lib/fpga_lab_system/fpga.html for detailed documentation, CAD drawings, etc of this system (also reachable via the course home page www.ce.chalmers.se/edu/course/EDA191). Its important that everybody studies these drawings (FPGA card, memory card, and monitor card).

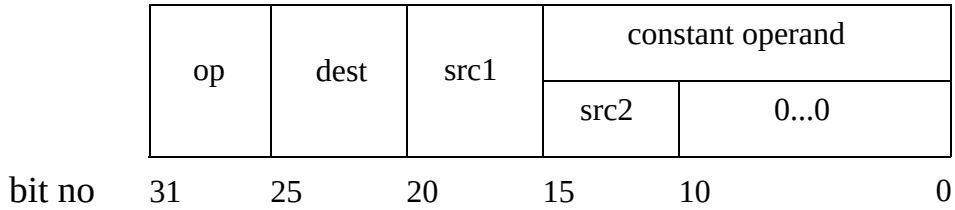
See also lab PM for description of the FPGA system and specification of the memory chips used.

Note! The Symfoni ISA uses byte-adresses. However, since the FPGA board doesn’t support byte adressing the address sent to memory must be a word-address (byte adress shifted 2 steps right).

1.2 Instruction Set and Encoding

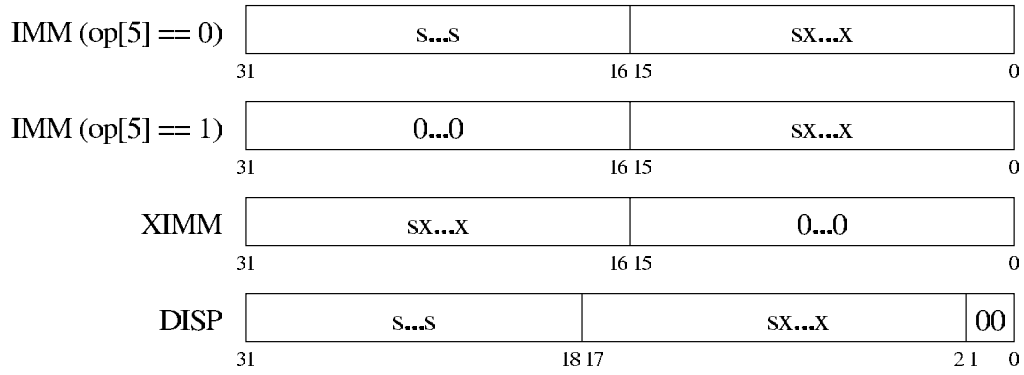
A Symfoni instruction is always one word. The different parts of the word are shown below in figure 1. There are three five bits register references: src1, src2 and dest. The op field holds the six bits op-code of the instruction.

FIGURE 1. Instruction Word Layout



Symfoni supports 22 different instructions. Each of these can be combined with one or more of the four different instruction modes: register (REG) that uses the src2 operand, immediate (IMM), extended immediate (XIMM) and displaced (DISP) that uses the constant operand. Figure 2 shows how the constant operand is extended from 16 to 32 bits in each mode and table 1 describes which instruction and instruction mode combinations that are available.

FIGURE 2. Immediate Format Interpretation



Note that ‘s’ indicates the most significant bit, holding the sign of the value, and ‘s...s’ indicates sign extension while ‘sx...x’ is the actual value.The instruction code encoding is shown in table 1. Please note that the different columns correspond to the different instruction modes available.

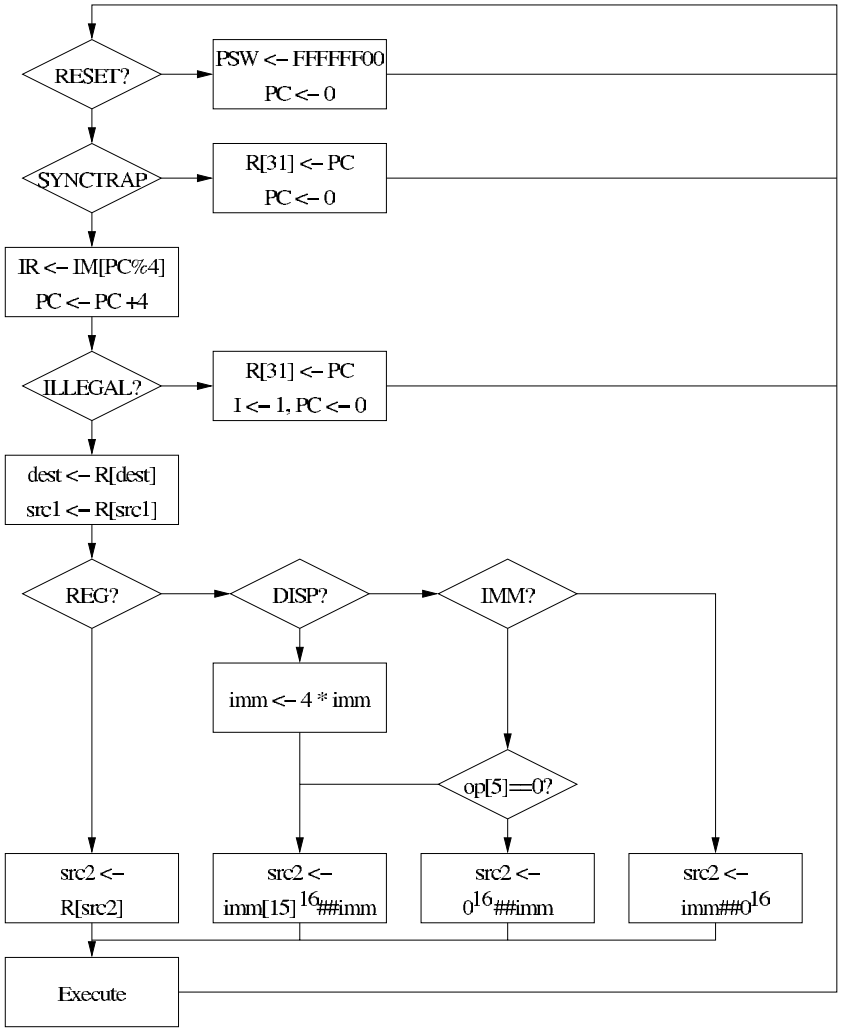
TABLE 1. Symfoni Instruction Set

op[1..0]	**00**	**01**	**10**	**11**
op[5..2]	**(REG)**	**(IMM)**	**(XIMM)**	**(DISP)**
0000	ADD	ADDI	ADDX	ADDD
0001	ADDV	ADDVI	ADDVX	ADDVD
0010	MUL	MULI		
0011	MULH	MULHI		
0100	SHZ	SHZI		BEQ
0101	SHS	SHSI		BNE
0110	CMP	CMPI	CMPX	CMPD
0111	JUMP		JUMPX	JUMPD
1000	SUB	GET		LW
1001	SUBV	PUT		SW
1010		TRAP		
1011	AND	ANDI	ANDX	
1100	OR	ORI	ORX	
1101	XOR	XORI	XORX	
1110	SET	SETI	SETX	
1111	RESET	RESETI	RESETX	

Note! the PUT and GET instructions are listed here in the ‘“IMM” column, but actually they are of the “REG” type.

The basic interpretation algorithm can be seen in figure 3. In table 2 the outcome of each instruction is specified in detail.

FIGURE 3. Interpretation Algorithm



The ## sign means concatenation. E.g. 0^{16}##imm means that sixteen zeroes are concatenated to the “imm” string.

The table below shows the exact meaning of each instruction.

TABLE 2. Detailed instruction function

Mnemonic	Meaning	Function
ADD[I|X|D]	Add	dest <- src1 + src2
ADDV[I|X|D]	Add and trap on overflow	if(overflow(src1 + src2) && V==0) { R[31] <- PC, PC <- 0, V <- 1 } else dest <- src1 + src2
SUB	Subtract	dest <- src1 - src2
SUBV	Subtract and trap on overflow	if(overflow(src1 - src2) && V==0) { R[31] <- PC, PC <- 0, V <- 1 } else dest <- src1 - src2
CMP[I|X|D]	Compare (less than)	dest <- (src1 < src2) ? 1 : 0
MUL[I]	Multiply and return LSW	dest <- (src1 * src2)[0..31]
MULH[I]	Multiply and return MSW	dest <- (src1 * src2)[32..63]
AND[I|X]	Bitwise AND	dest <- src1 & src2
OR[I|X]	Bitwise OR	dest <- src1 | src2
XOR[I|X]	Bitwise XOR	dest <- scr1 ^ src2
RESET[I|X]	Read and reset status bits	dest <- PSW, PSW <- PSW & !src2
SET[I|X]	Read and set status bit	dest <- PSW, PSW <- PSW | src2
SHZ[I]	Shift in zero from left or right	dest <- (src2<0)? (0...0##src1>>-src2) : (src1##0...0<<src2)
SHS[I]	Shift in sign from left or right	dest <- (src2<0)? (src1[31]...src1[31]##src1>>-src2) : (src1##src1[31]...src1[31]<<src2
LW	Load word form data memory	dest <- DM((src1 + src2) / 4)
SW	Store word to data memory	DM((src1 + src2) / 4) <- dest
GET	Get data from IN port	dest <- IN, PSW <- PSW & !src2
PUT	Put data on OUT port	OUT <- src1, PSW <- PSW | src2
BEQ	Branch if equal	PC <- (dest == scr1) ? (PC+src2) : PC
BNE	Branch if not equal	PC <- (dest != src1) ? (PC+src2) : PC
JUMP[X|D]	Jump	dest <- PC, PC <- src1 + src2
TRAP	Jump to trap handler	R[31] <- PC, PC <- 0, PSW <- PSW | src2

Notice that "/ 4" indicates that the addresses used to address PM & DM are word addresses (where each location is 32 bits wide).

Note also that the "src2" field refer to a register in the REG type and to an immediate field otherwise.

The branch instructions (BEQ and BNE) does not require three source registers as these instructions are in the DISP column of table 1 and the src2 register will therefor be a constant value.

1.2.1 Traps, Interrupts and Initialisation

After having received a RESET signal the processor starts in a predefined state. The PC is set to zero, and PSW to $FFFFFF00_{16}$ when the fetching and execution of instructions are started.

When the SYNCTRAP signal is raised or an overflow or illegal instruction is detected, a hardware interrupt occurs. When a hardware interrupt or a software trap occurs the PSW is setup to indicate the source of the trap/interrupt and the execution is resumed from PC=0. The address of the interrupted instruction is stored in R31.

1.3 Programming Model

1.3.1 Data types

The basic word used in Symfoni is 32 bits wide. The bits are numbered 0 to 31 from the least significant bit to the most significant bit.

1.3.2 Visible State Variables

Both the instruction memory (IM) and the data memory (DM) (IM & DM are described further down) have the address space $0..2^{32}-1$, thus one address can be held in one word. All addresses in IM and DM refer to a byte, but both IM and DM can only access aligned words (i.e. the two LSBs are assumed to be zero). The instruction memory is used to hold instructions and the data memory is used to hold data.

The register file, R, consists of 32 registers. Register 0 (zero) always holds the value zero. Registers 1 to 31 are general purpose and can be used to hold any value. Register 31 is used to hold the return address from traps and interrupts, but can be used to hold any value if this is taken under consideration.

The program counter, PC, always points to the instruction to be fetched and executed. It is used to address the IM, and is thus assumed to be word aligned. If the instruction at address n is being executed then the PC normally points at the address n+4. However, the PC is altered when jumps, traps and branches are executed.

The input port, IN, is a 32 bits wide register holding data from Symfoni's external port. The output port, OUT, is also a 32 bits wide register holding data to be sent to Symfoni's external port.

The processor status word, PSW, is a 32 bits wide register holding status information concerning external communications, interrupts and the timing or the periodic interrupt generator. The contents of the bits are described in table 3.

.

TABLE 3. PSW Contents

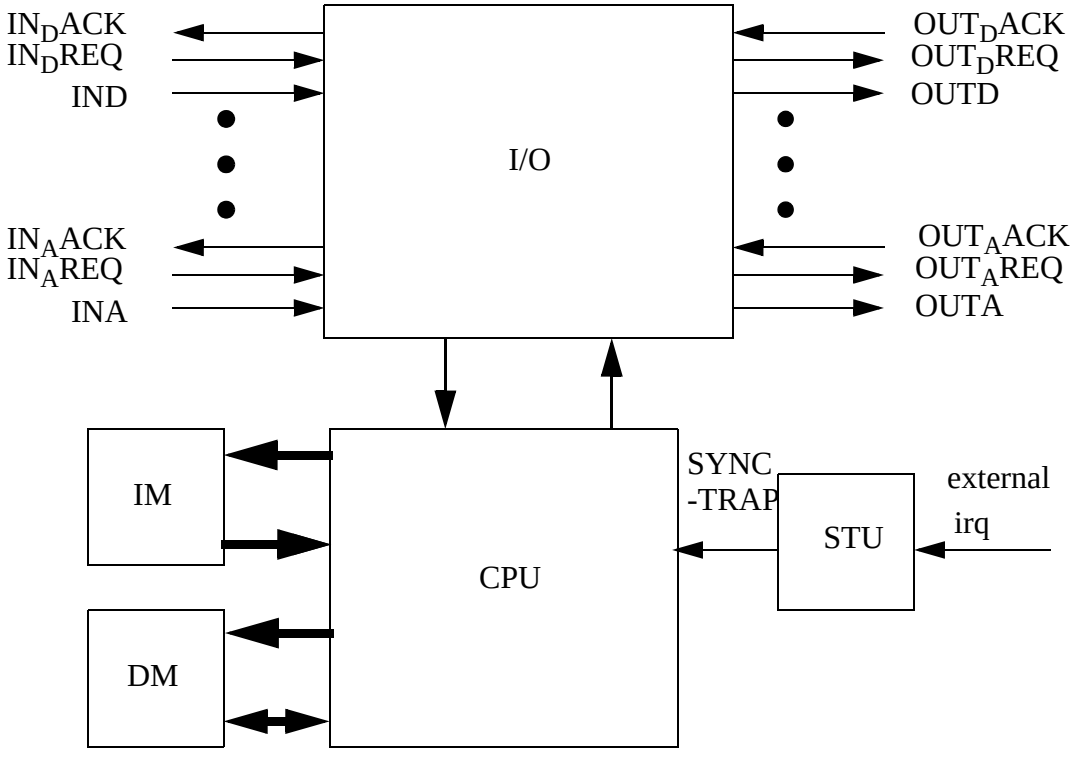
Bit(s)	Name	Meaning
0..3	IB[0..3]	Input port byte available flags
4..7	OB[0..3]	Output port byte available flags.
8	V	Arithmetic overflow flag (internal)
9	I	Illegal instruction flag (internal)
10	reserved	Reserved for future extensions (internal)
11..14	X[0..3]	External interrupt flags/user defined trap flags
15	T	Timer interrupt flags
16..31	TP	Timer period (divided by 1024 CLK cycles)

2.0 System Overview

The Symfoni system architecture is shown in figure 4. It features four units outside the CPU core, the Timer and Synchronisation Unit, STU, the I/O unit and the two memories, IM and DM. The I/O unit implements asynchronous I/O using handshaking signals.

The groups are also encouraged to investigate the possibility of using the keyboard and lcd display as additional I/Os. This is one out of three “extra” assignments in the project that can be chosen if the group so decides (see further down).

FIGURE 4. System Overview



2.1 The Timer and Synchronisation Unit

The STU manages the SYNCTRAP signal and the interrupt timer. The SYNCTRAP signal is used to indicate that an external interrupt or a timer interrupt has occurred. This is indicated by setting the signal high during one clock cycle.

The interrupt timer is a 26 bits counter. The counter decreases its value by one each clock cycle. When it reaches zero it checks the value of the bit T in PSW (see table 3 for a description of the bits of PSW). If T is zero an interrupt is generated through SYNCTRAP, and T is set to one in PSW. If T already is one, no interrupt occurs. When it reaches zero the counter is reloaded with the value TP from PSW in the most significant bits, and ten zeroes in the least significant bits.

Note! The STU unit must be constructed by the group, it does not exist on the board.

2.2 The I/O Unit

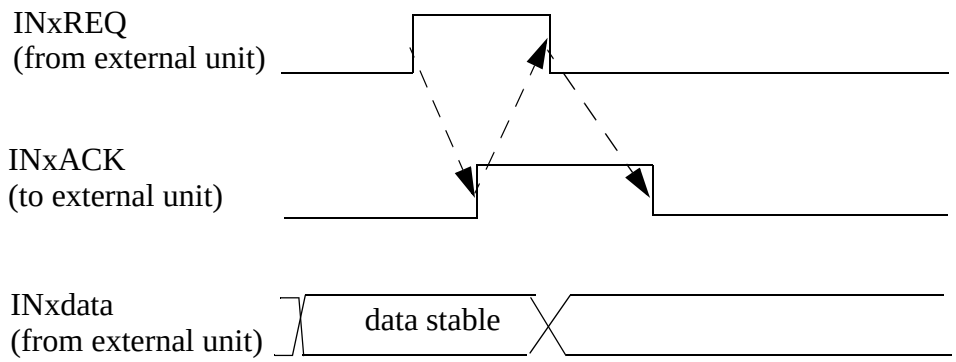
The communications with the outer world is managed through two 32 bits wide registers: IN and OUT. All communications are controlled by a handshaking protocol using the signals IN_xREQ and IN_xACK, where x=A,B,C,D.

IN stores data gathered from four external busses, INA-D (INA=$IN_{0..7}$, INB=$IN_{8..15}$, etc.) For each group a status bit, IB_x, is held in PSW (see table 3 , bits 0 .. 3) that indicates that new data is available (x=A,B,C,D). These bits are set as soon as a byte has been received. As long as the bit is set, no new data is accepted from the outside.

OUT stores the data to be exported on the external busses OUTA-D (mapped as INA-B to the OUT register). Each byte has a status bit, OB_x, in PSW (see table 3 , bits 4..7) indicating when the corresponding byte is free for new data in the OUT register. When a byte has been transmitted, the corresponding bit is set to zero.

The handshaking protocol is as follows (“full handshake”):

FIGURE 5. Asynchronous handshaking protocl



I

The 32-bit “IN” register is used to temporarily hold received data from the INx ports. When a byte has been received and loaded into IN, INxACK should be set and the corresponding PSW bit set by the hardware. When the cpu has read this byte from the IN register (by a GET instr), the PSW bit should be cleared. INxACK should be cleared when INxREQ goes low.

Analogous for the out-ports, OUTx, x=A,B,C,D and the “OUT” register. The PSW bits (4..7) indicates when the corresponding byte is free in the OUT register.

Note! The IO unit is a part of the CPU and must be constructed by the group, it does not exist on the board.

Also, to test this unit (using testprogram P2) a block outside the CPU (but still inside the FPGA), should be constructed that act as a “dummy” external I/O unit.

2.3 Memory

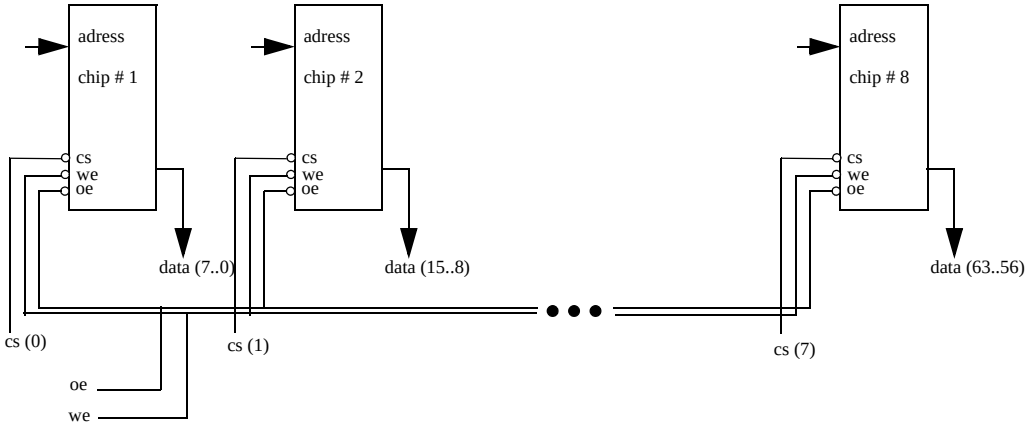
The memory on the lab system consists of two SRAM arrays with 8 KM68100C-15 ICs, each maximum 512k x 8 bit large. The arrays are connected to form two 512k x 64 bit SRAM units, IM and DM. They are accessed by the FPGA (CPU) through an 20 bits wide address bus (maximum) and a 64 bits wide data bus.

Note! the 512k size figure is the maximum value, typically the FPGA system memory board only uses 128k modules.

Every single SRAM IC on the system can be selected or deselected with a Chip Select, CS, signal and the two whole arrays with Output Enable, OE, and Write Enable, WE, signals. These signals are together referred to as the control bus. See figure 6 below.

The indivudual CS signals make it possible to use a 8,16,24,32,40,48,56, or 64 bits wide databus. Since the CPU uses 32 bit words, only half of the memory width is sufficient to use in Symfoni.

FIGURE 6. Memory circuits



Since two memory arrays are present on the board, this makes it possible to design the Symfoni processor as a Harvard architecture (IM and DM memories).

You should refer to the appended specification in the course lab PM for complete information of the memory chips.

3.0 Clock speed optimization

To estimate the pipeline clock speed, synthesis has to be made. This should be started early in the design process to get estimates on the various pipeline stage signal delays. Using this information you can try and speed up the critical path by e.g. deeper pipelining or by moving blocks between pipe stages. Remember also that the overall performance must be evaluated and optimized using the testprograms P0,P1, and P2 and the formula T_{exe}=IC*CPI*T_c.

4.0 Symfoni assembler

There is an assembler available for the Symfoni ISA instruction set. It is a Perl script that run on Linux and Unix using Unix file types. It can be found at www.ce.chalmers.se\edu\course\EDA191\Symfoniassembler or via the course home page.

5.0 Additional assignments

If time permits, the following assignments can be chosen.

5.1 Cache

On-chip cache (tag-part and data-part). Direct mapped, set-associative or fully associative. This is in a real situation really only viable if the memory speed is slower than the pipeline clock speed. So, groups choosing this option should first try and optimize clock speed, then goto cache design and implementation.

5.2 Floating point

Design a floating point unit that implements some floating point instructions (e.g. FL-ADD, FL_SUB, FL_LOAD, FL_MUL),.

5.3 Additional I/O

Using the keyboard and the LCD display on the FPGA board as additional I/O (apart from the asynhronous interface).