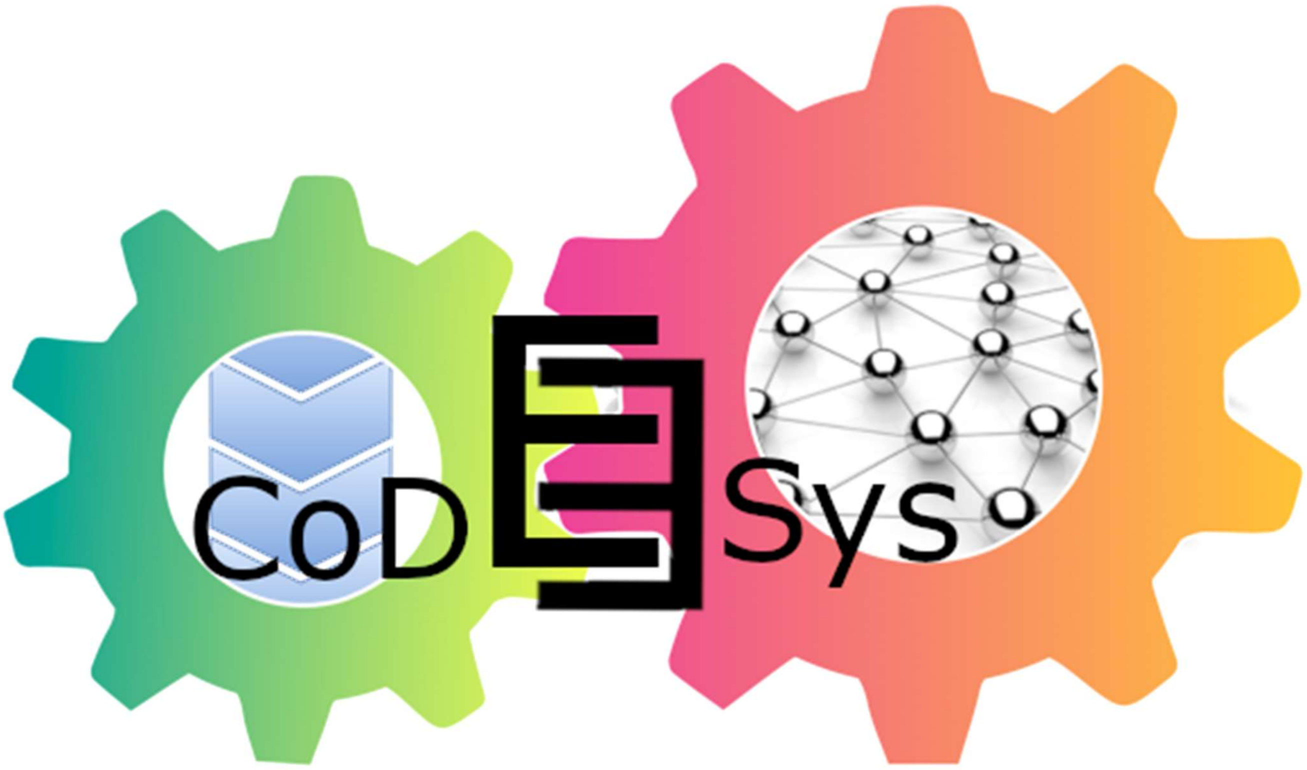
Secapem
targets and scoring systems

Jan 2019 V1.05



Codesign Embedded System

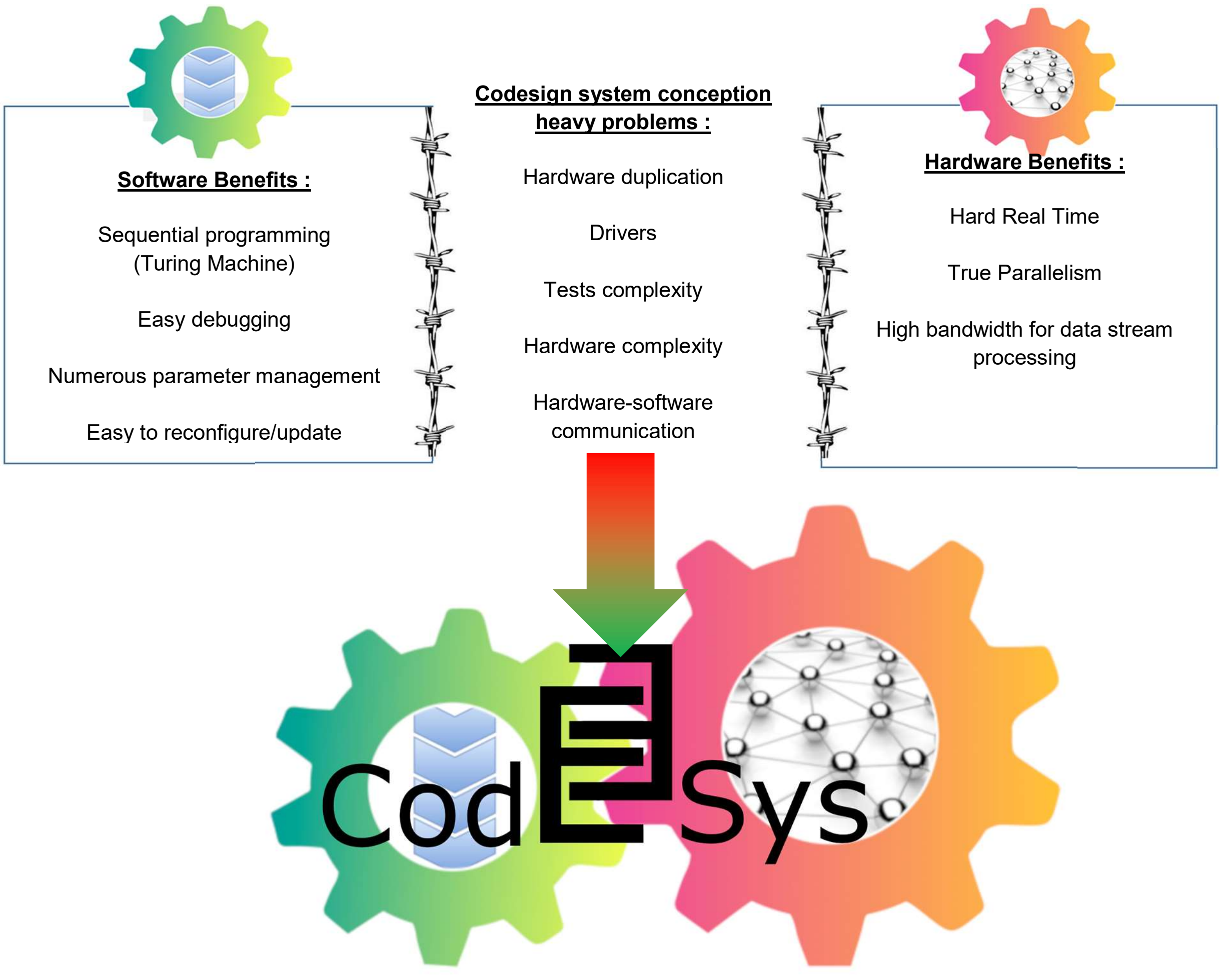
1. TABLE OF CONTENTS

CodESys

Codesign **E**mbedded **Sys**tem

2. SOFTWARE/HARDWARE CODESIGN

Software systems based on Turing/Von Neumann machines such as processor or microcontrollers. They are based on a CPU (Central Processing Unit) which executes sequentially a simple instruction. Their main benefits are to be easy to program, to provide a high flexibility of reprograming and to be able to execute any complex operation. The first disadvantage of these systems is to execute only one task at time. The second disadvantage is to segment an operation into a set of simple


Software Benefits :
Sequential programming (Turing Machine)
Easy debugging
Numerous parameter management
Easy to reconfigure/update
Codesign system conception heavy problems :
Hardware duplication
Drivers
Tests complexity
Hardware complexity
Hardware-software communication
Hardware Benefits :
Hard Real Time
True Parallelism
High bandwidth for data stream processing
CodESys

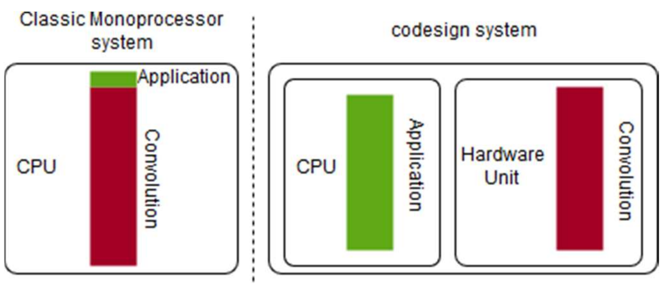
instructions which slows down the execution of the operation. Also, if the system uses an operating system, it is difficult to predict how much time the operation will take (furthermore if the system has several levels of memory cache).

In hardware systems the operation to execute is directly wired in hardware component such as an ASIC or a FPGA. The operation is executed very quickly because it is directly implemented with electronic component. Also, if the operator is wired several times in the component, several operations can be executed at same time. Because the implementation doesn't depend on instruction set or operating system it is much easier to predict the execution time of an operation. But describing hardware architecture in VHDL or Verilog description language is a little bit more complicated than using a programming language for software systems. The signal propagation time, the clock synchronisation, the critical path (etc…) require some experience. Also, when hardware component is configured to execute an operation it is harder to reconfigure it than reprogramming a software system. Without entering into details, the debug process of a hardware system requires also some equipment (scopes, logic analyser, etc.) and experience.

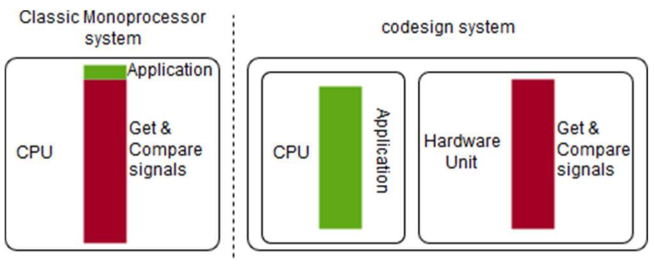
These two kinds of systems have complementary benefits and disadvantages. That's why the idea to design systems which merge soft and hard operators has been born (developed). This kind of mixed design is called « hardware-software codesign »: software to keep advantage of programming languages (flexibility, reprogramming, etc..) and hardware to take benefit from high speed execution and time predictability.

The conception of a codesign system is complex and brings a lot of problems which often eliminate this kind of solution. The most important problems are : the communication between the software and the hardware parts, the tests

A simple example could be a digital signal processing algorithm which often needs the convolution operator. This operator is very time consuming for a processor. In a codesign system you can create your convolution operator in hardware and the rest of the algorithm in software. First, the both parts could work in parallel and second the execution of the convolution will be much faster than if it was programmed in software.

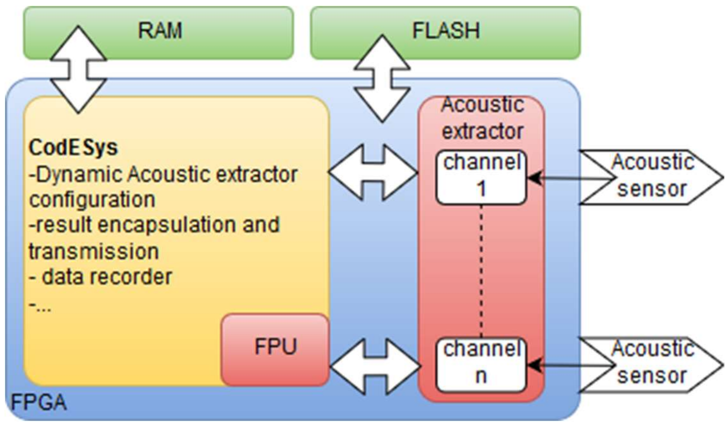


Another example: you want to execute a specific task when some analog input signals sampled at 1MHz reach a specified combination (for example: signal S1=1V s2=2.3V, s3<4V, etc…). In this case the microcontroller can only execute this task (get signals and compare with the specified combination) without doing anything else. In a codesign system, a specific hardware unit can be designed to get signals



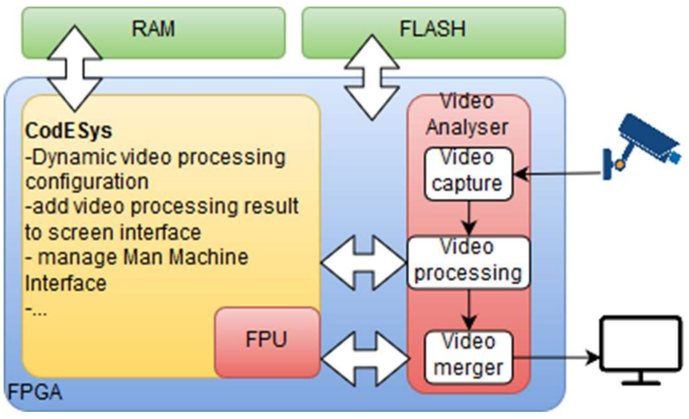
and compare with a specified combination and interrupt the processor when the combination is reached. In this case the processor is freed from the « watching » task and is able to work on the application.

<u>Some systems made with CodESys</u>:

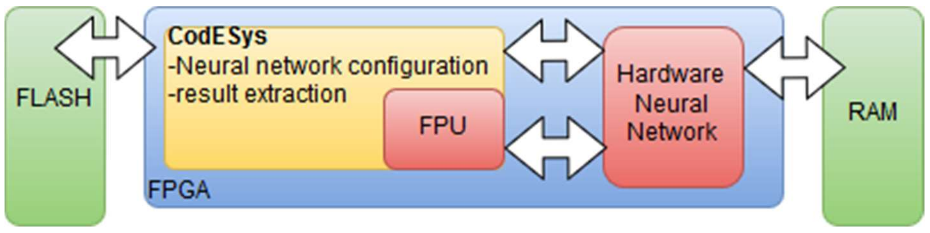


Secapem embedded scoring system

Video processing and overlaying application


RAM
FLASH
CodESys
-Dynamic video processing configuration
-add video processing result to screen interface
- manage Man Machine Interface
-...
FPU
FPGA
Video Analyser
Video capture
Video processing
Video merger

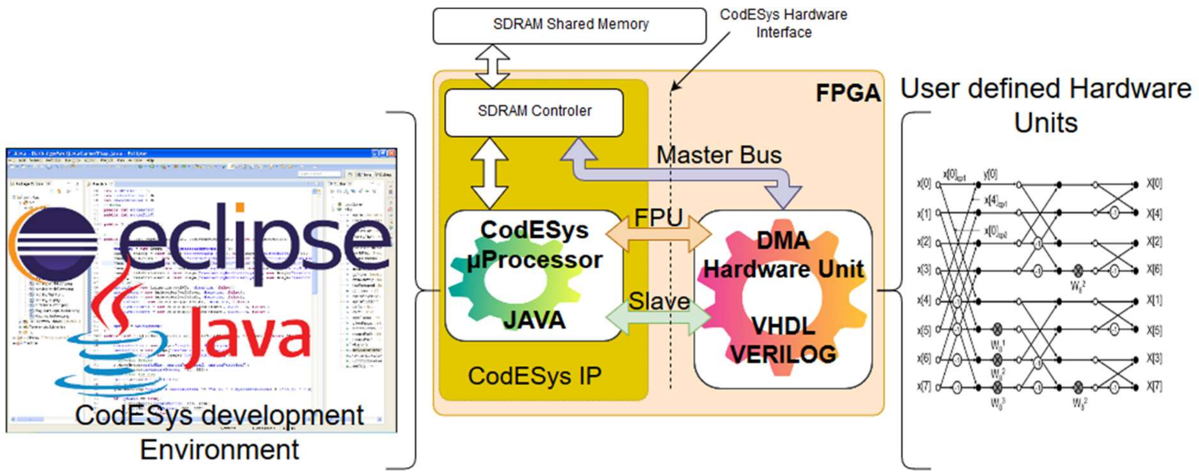
Neural processor :



3. OVERVIEW

We have created CoDESys in order to simplify the design of hardware/software codesign embedded systems. To reduce size and complexity of electronic system all units (hardware and software) are gathered in a single FPGA. CoDESys provides:

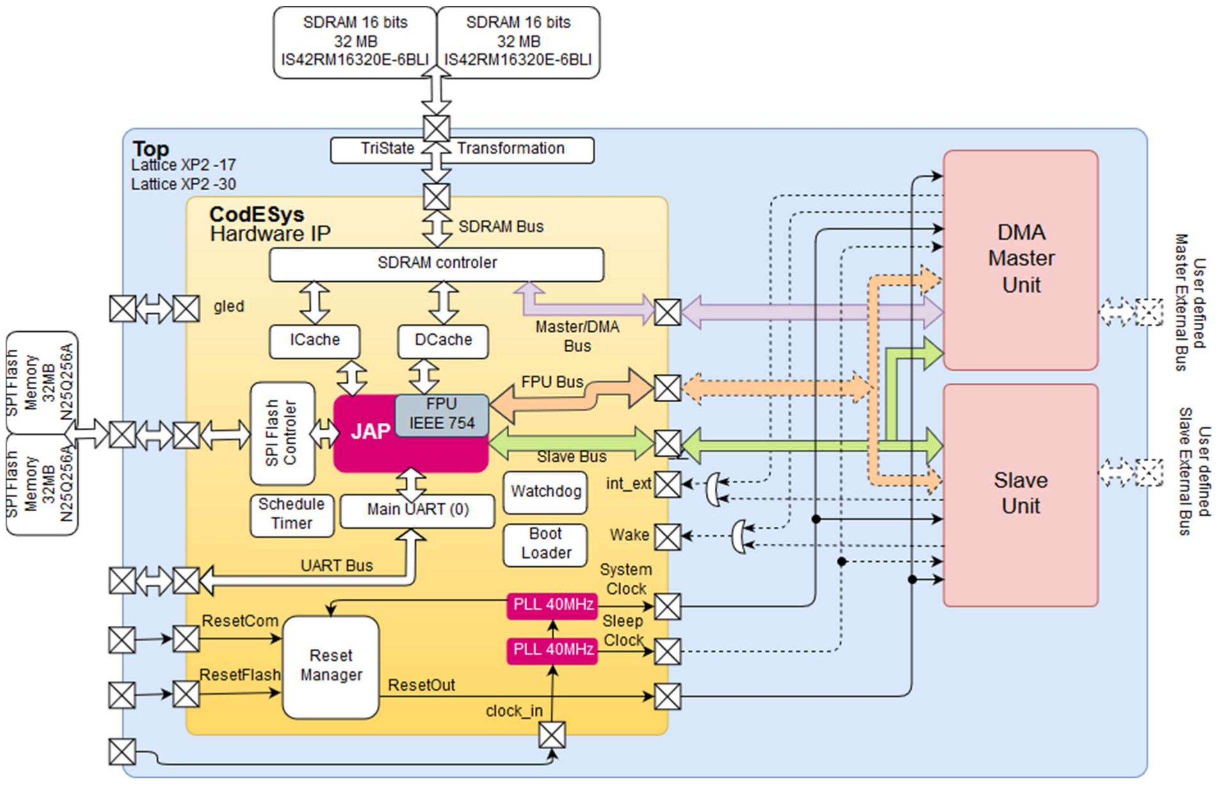
- a programming language (Java)
- a programming environment (Eclipse)
- a processor (software system)
- a shared access to the IEEE-754 FPU
- a shared access to the SDRAM (MASTER bus)
- a processor access to the hardware units (SLAVE bus)
- An API to communicate with hardware unit(s).


SDRAM Shared Memory
CodESys Hardware Interface
FPGA
User defined Hardware Units
SDRAM Controler
Master Bus
FPU
Slave
CodESys µProcessor JAVA
DMA Hardware Unit VHDL VERILOG
CodESys IP
eclipse
Java
CodESys development Environment

The SLAVE BUS is used to configure the hardware unit registers. In this case the hardware unit acts as a slave and only respond to the processor requests.

In order to be totally autonomous and able to work in parallel with the processor, hardware units have a direct access to memory (DMA: Direct Memory Access). They can work on big amount of data without interrupting the processor which have its own memory caches. In this case the hardware unit acts as a master and is able to request some reads or writes to the SDRAM memory. This bus is called the MASTER BUS.

4. CODESYS HARDWARE ARCHITECTURE



The CodESys IP includes :

- A SDRAM controller: allow the processor and the DMA master unit to load/store from/into SDRAM memory
- A SPI Flash controller to communicate with the non-volatile Flash memory
- A reset manager (explained further)
- A Schedule Timer used by the Java Virtual Machine to schedule Java Threads
- An UART: used to update software/firmware or download/upload data
- A clock manager (explained further)
- A bootloader: the CodESys can boot on UART or on Flash memory. It boots on UART when the application or firmware must be updated (update mode). It boots on Flash to run the application (normal use mode).
- Memory cache: fast internal memory cache to speed up the execution
- watchdog: hardware device which resets de CodESys when application is stuck somewhere.
- JAP: Java Virtual Machine including software part and hardware acceleration. It includes **an IEEE 754 floating Point Unit.**
- Master DMA Bus : bus used to connect a master hardware unit to the SDRAM (explained further)
- Slave DMA Bus: bus used to connect a master/slave hardware unit
- FPU bus: bus used to access the IEEE 754 floating point unit. In this case the FPU is shared with the processor.

- Sleep_clock : This clock still works even if the CodESys IP is in sleep mode. It must be used by hardware unit which are not sleeping in sleep mode (usually the hardware unit which are able to wake up the system with WAKE signal).
- System_clock : this clock must be used to communicate with SLAVE and MASTER buses.
- Wake signal : this signal is used to wake up the system.
- Int_ext : this signal is used to send an interrupt to CodESys.

4.1 CodESys processor

The CodEsys processor has two instruction sets: Java and Risc. This processor has been designed with in collaboration with the CEA-DAM. It is working in military/industrial applications for twenty years. The Java has been chosen for its sustainability, security and "all inclusive" implementation. Java includes all mechanisms of an operating system, such as memory allocation/free, threads, thread synchronisation, object oriented, etc.

4.2 Clock system:

Clock is generated from an external quartz of 40MHz. This external clock must be connected to clock_in signal of CodESys. This clock is synchronized into PLLs and provided as clock_out signal. The frequency of clock_out signal is 40MHz. The clock_out signal must be used with all provided buses (master, slave and FPU).

All signals are synchronized on the rising_edge of clock_out.

A sleep_clock is also provided by CodESys IP. The CodESys IP can be set (by software) in sleep mode. When the CodESys IP is in sleep mode, the clock_out signal is no longer generated in order to freeze all hardware units which doesn't need to work when sleeping. But some units can be used as wake system and need a clock signal. These units must use sleep_clock because the sleep_clock still works even if CodESys is in sleep mode. The frequency of sleep_clock is 40MHz.

4.3 Reset system:

CodESys has two external input reset signal : reset_fla and reset_com.

Reset_fla resets CodESys and sets it in Flash boot mode (normal use). So reset_fla signal is the "normal" reset.

Reset_com resets CodEsys and sets it in COM(UART) boot mode (update). Reset_com must be used when an application or firmware update is needed.

Reset_out is generated either when reset_fla or reset_com occurs but it takes also into account the synchronisation of the PLLs with the external clock (clock_in). reset_out must be used to reset all hardware units.

All reset signals are active low.

4.4 GLED signal:

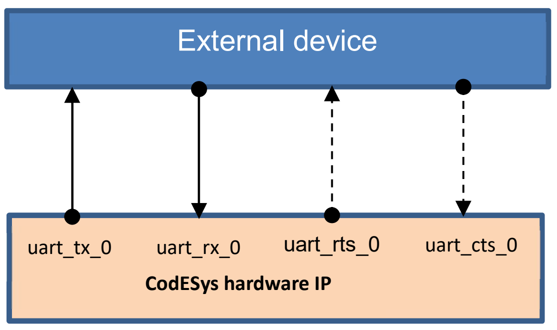
This signal is used to see if the system is working correctly. In application mode (normal use) this signal toggle approximatively every second.

When updating the system, the led toggles when a data is received.

When this signal stays unchanged, it means the system is stuck somewhere.

4.5 UART Bus :

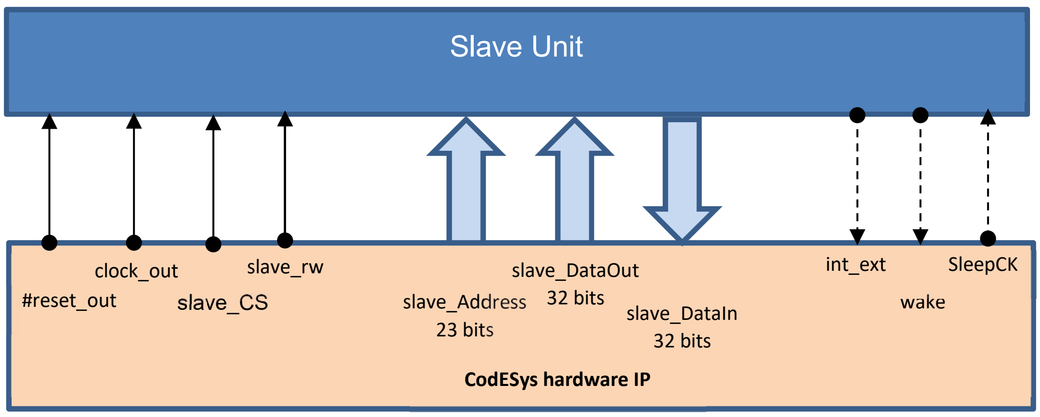
The CodESys uses the UART bus for two main goals. First in update mode, it is used to update application and firmware. Second in running mode (application) it can be used to communicate with an UART device (PC, module, etc).



It is a standard TTL UART. The default settings are:

115200 bauds, 1 stop bit, no parity

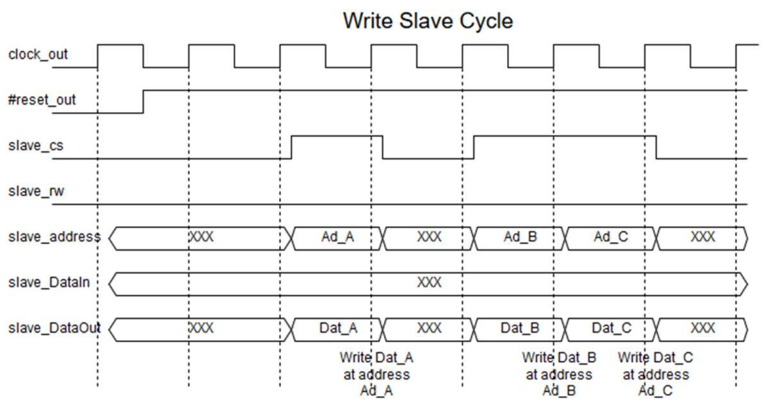
4.6 Slave Bus



SLAVE bus is used by the processor to read or write data into hardware units.

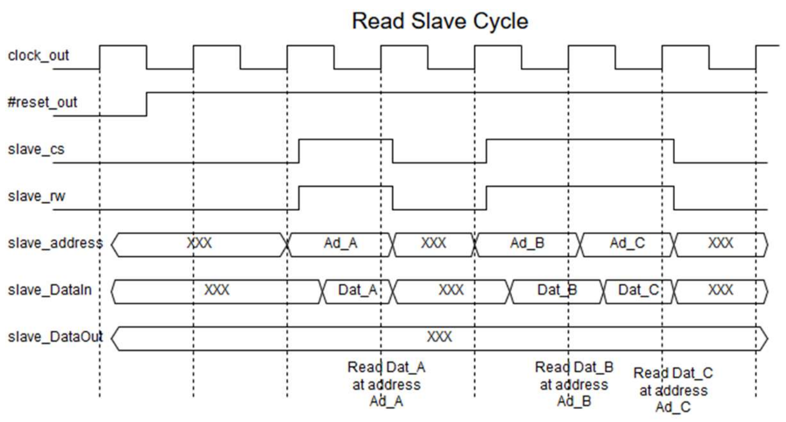
4.6.1 SLAVE Write Access

When the processor wants to write a data into a specific register, it resets the slave_RW signal, sets the slave_cs, sets the



slave_address bus and sets the slave_DataOut bus. The slave_cs signal means the processor wants select a hardware unit register. The slave_rw low means a write cycle. The slave_address bus specifies the address of the requested register. The slave_DataOut bus specifies the value of the data the processor wants to write in the register. The slave unit must update the specified register at the rising edge of the clock_out signal.

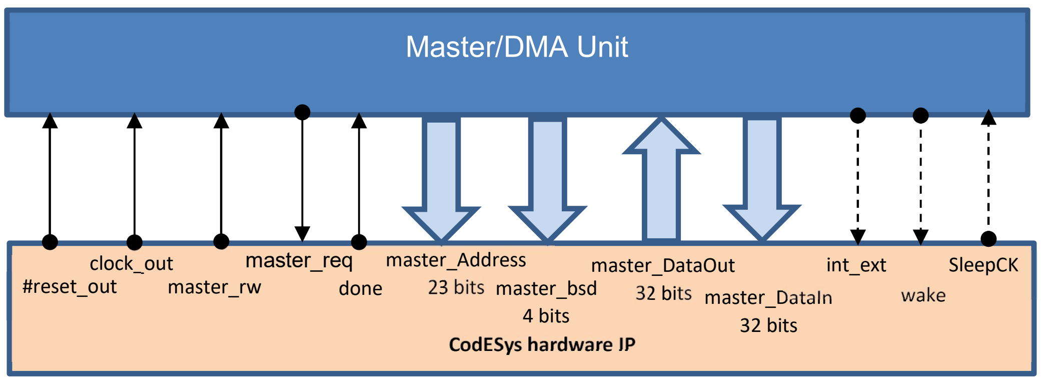
4.6.2 SLAVE Read Access



When the processor wants to read a data from a specific register, it sets the slave_cs , sets the slave_rw signal and sets the slave_address bus. The slave_cs signal means the processor wants to select a hardware unit register. The slave_rw low means a write cycle. The slave_address bus specifies the address of the requested register. Before the next rising edge of clock_out the selected hardware unit must set the slave_DataIn bus with the value of the selected register. The data is transferred to the processor on the rising edge of clock_out signal.

4.7 Master Bus

The master bus is used by hardware units to read and write into memory. This bus is also called a DMA (Direct Memory Access) bus. This kind of bus makes the hardware units totally autonomous and they can work on a lot of data without the processor help. Thus processor and hardware units are able to work in parallel.

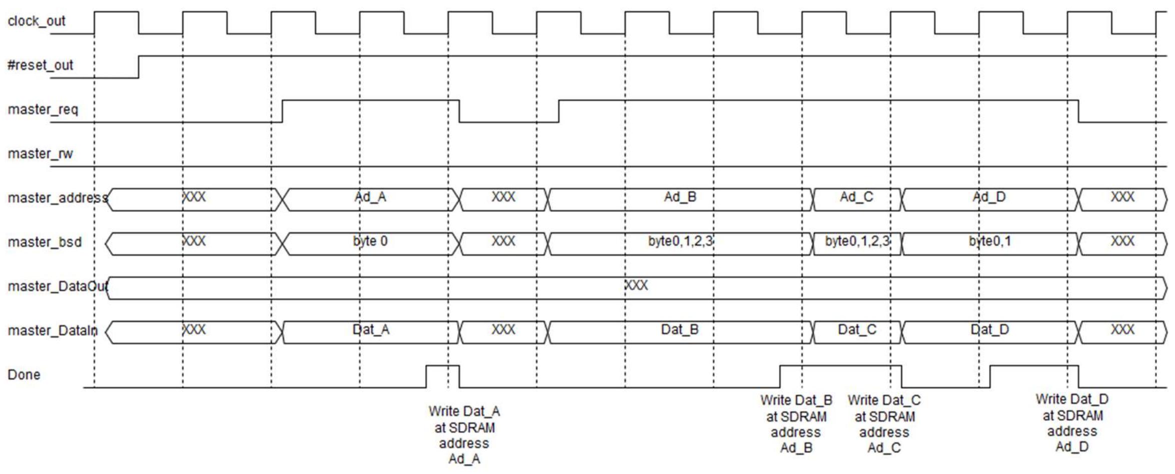


The processor and the hardware units share the same SDRAM memory. The hardware units have a higher priority to the processor. When the processor and the hardware unit want to access the SDRAM at the same time, the SDRAM controller freezes the processor and give access to the hardware unit. The processor doesn't take a lot of SRAM bandwidth because it has its own memory caches: One for instruction and one for data (Harvard architecture).

4.7.1 Master Write Cycle

When a hardware unit wants to write a data into SDRAM, it sets master_req signal, master_rw, master_address bus, master_bsd and master_DataIn. master_req sends a request to the SDRAM controller. master_rw low means a write cycle. master_address specifies the SDRAM address to write. master_address is a 32 bits (4 bytes) address. master_bsd specifies the byte(s) into the 4 bytes word. For example if master_address=A and bsd(0)=1, bsd(1)=0, bsd(2)=0 and bsd(3)=0, only the byte at SDRAM address A*4 will be written. If master_address=A and bsd(0)=1, bsd(1)=0, bsd(2)=1 and bsd(3)=0, only the bytes at SDRAM address A*4 and (A*4)+2 will be written. If we want to write a 4 bytes word at SDRAM address 8 we have to set master_address=2 and bsd(0)=1, bsd(1)=1, bsd(2)=1, bsd(3)=1. If we want to write only one byte at SDRAM address 17 we must set master_address=4 and bsd(0)=0, bsd(1)=1, bsd(2)=0 and bsd(3)=0. The data to write must be in the 32 bits master_DataIn Bus.

Write Master Cycle (DMA cycle)



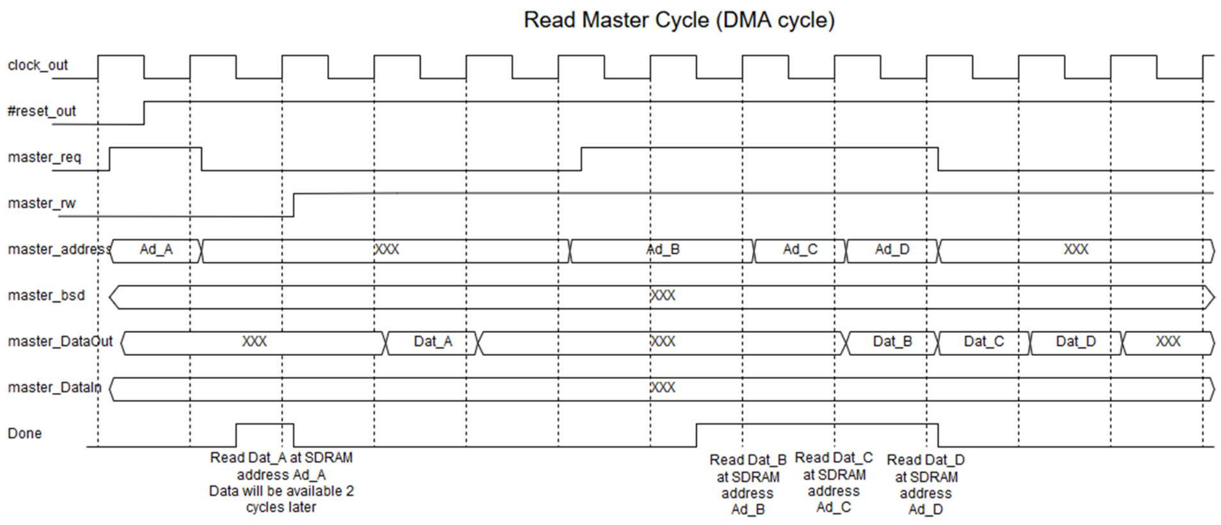
Master_address	bsd	SDRAM byte address
A	bsd(0)=1	A*4
	bsd(1)=1	(A*4)+1
	bsd(2)=1	(A*4)+2
	bsd(3)=1	(A*4)+3

The data is written into memory when done is set to one. Done signal can be set to one from 1 to n cycles.

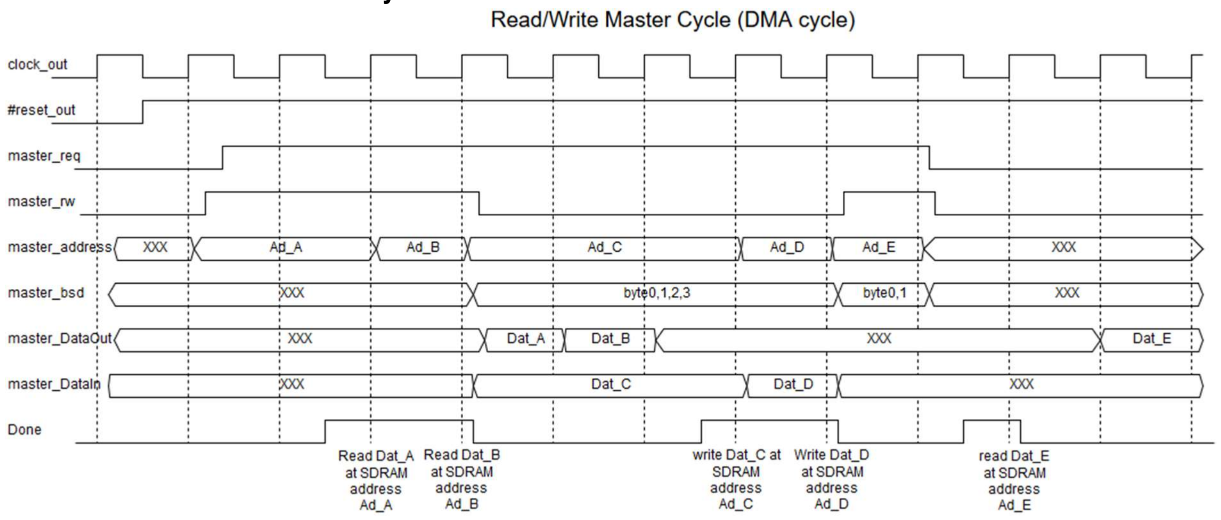
Several write cycle can be chained. In this case one data write can be done each clock cycle (burst mode). Warning: the done signal can be lowered during some cycles (for example during refresh memory cycles).

4.7.2 Master Read Cycle

When a hardware unit wants to read a data from SDRAM, it sets master_req signal, sets master_rw and master_address bus. No need to set master_bsd because the master bus always read a word of 32 bits. master_req sends a request to the SDRAM controller. master_rw high means a read cycle. master_address specifies the SDRAM address to read. master_address is a 32 bits (4 bytes) address. The done signal is set by CodESys to inform that request has been taken into account but the data will be available at least 2 clock cycles after. The data is ready on master_DataOut bus two clock cycles after the done signal. As write cycle, read cycle can be executed in burst mode (data is always delayed by two clock cycles).


Read Master Cycle (DMA cycle)
clock_out
#reset_out
master_req
master_rw
master_address
Ad_A
XXX
Ad_B
Ad_C
Ad_D
XXX
master_bsd
XXX
master_DataOut
XXX
Dat_A
XXX
Dat_B
Dat_C
Dat_D
XXX
master_DataIn
XXX
Done
Read Dat_A at SDRAM address Ad_A Data will be available 2 cycles later
Read Dat_B at SDRAM address Ad_B
Read Dat_C at SDRAM address Ad_C
Read Dat_D at SDRAM address Ad_D

4.7.3 MASTER READ/WRITE cycles


Read/Write Master Cycle (DMA cycle)
clock_out
#reset_out
master_req
master_rw
master_address
master_bsd
master_DataOut
master_DataIn
Done
XXX
Ad_A
Ad_B
Ad_C
Ad_D
Ad_E
byte0,1,2,3
byte0,1
Dat_A
Dat_B
Dat_C
Dat_D
Dat_E
Read Dat_A at SDRAM address Ad_A
Read Dat_B at SDRAM address Ad_B
write Dat_C at SDRAM address Ad_C
Write Dat_D at SDRAM address Ad_D
read Dat_E at SDRAM address Ad_E

A read cycle can be followed by a write cycle without lowering the master_req signal. When a write cycles follows a read cycle it is been delayed at least by two clock cycles.

4.8 FPU bus

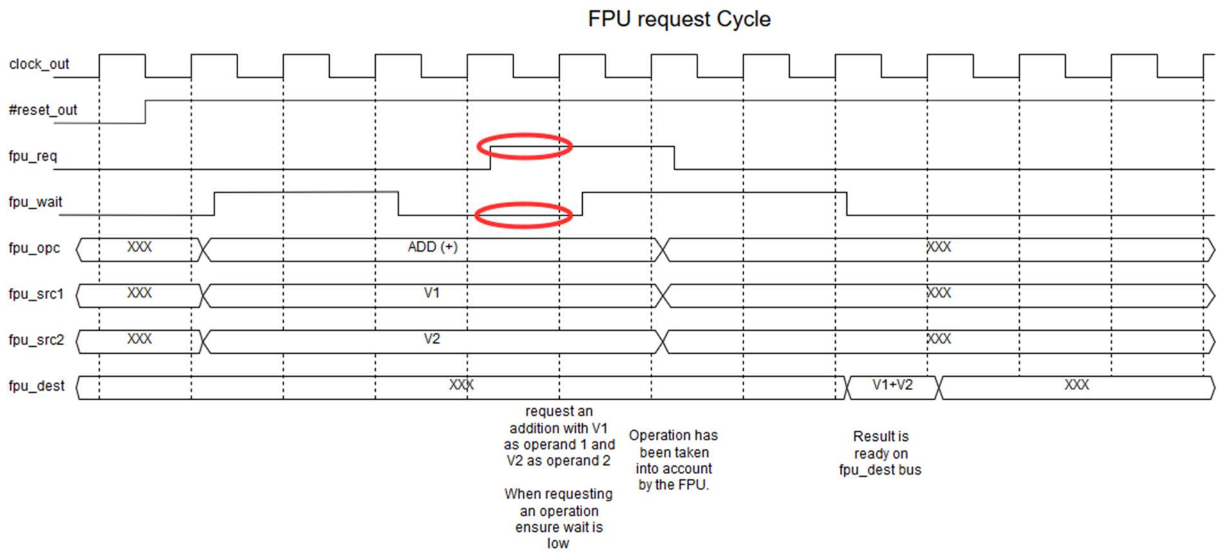
The processor includes an IEEE 754 Floating Point Unit. This FPU can be shared between the processor and with the hardware units in order to allow floating point processing.

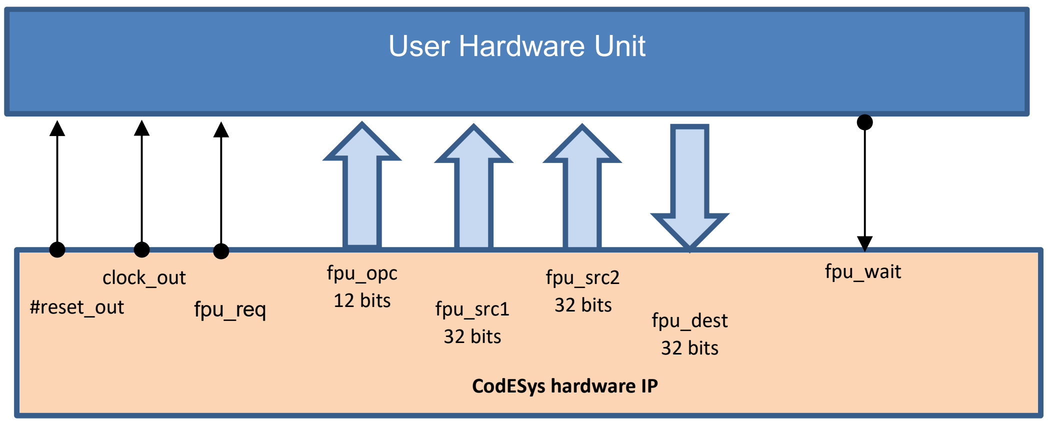
In order to request the FPU for an operation the hardware unit must set fpu_opc, fpu_src1, fpu_src2 and fpu_req. fpu_opc specifies the operation to perform.

IMPORTANT: In order to set fpu_req, you must ensure that the "wait" signal is low. If wait signal is high, it means the processor is using it.

The operation is taken into account when wait signal is set to 1.

The result is ready on fpu_dest when wait signal is low.


FPU request Cycle
clock_out
#reset_out
fpu_req
fpu_wait
fpu_opc
fpu_src1
fpu_src2
fpu_dest
XXX
ADD (+)
V1
V2
V1+V2
request an addition with V1 as operand 1 and V2 as operand 2
When requesting an operation ensure wait is low
Operation has been taken into account by the FPU.
Result is ready on fpu_dest bus



4.9 CodESys Interrupt

Hardware units can send interrupt to the processor by setting *int_ext* to one during one clock cycle. When CodESys interrupt manager receives the external interrupt which is the most important priority, the processor stops its current tasks and executes the CodESys.ITHandler() method. When the method is finished the processor resumes the current task.

5. CODESYS SOFTWARE API

5.1 SLAVE API

5.1.1 SLAVE write method

public static void slaveWrite(int ad, int value)

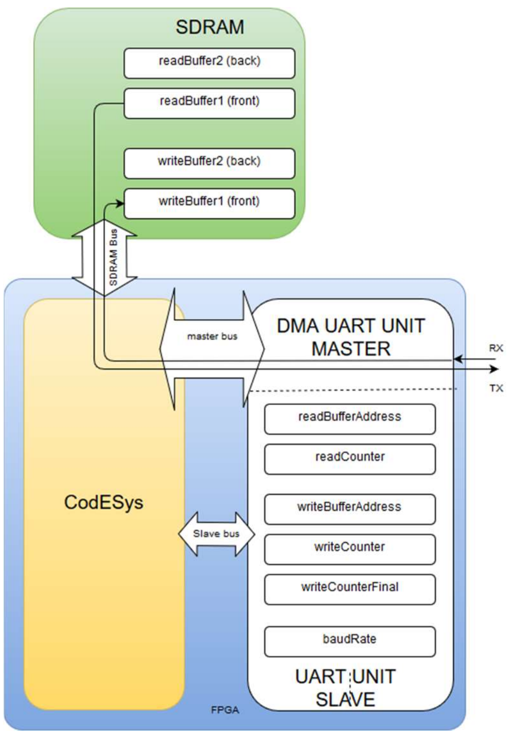
This method makes a write access on slave bus. It puts *ad* on the slave_address bus, *value* on data_out bus, resets slave_rw and sets slave_cs.

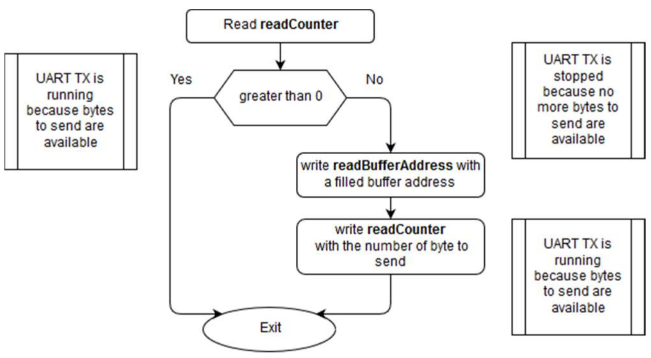
5.1.2 SLAVE Read method

public static int slaveRead(int ad)

This method makes a read access on slave bus. It puts *ad* on the slave_address bus, set slave_rw and slave_cs and returns the value read on the data_in bus.

5.2 MASTER DMA API




Read readCounter
greater than 0
Yes
No
UART TX is running because bytes to send are available
write readBufferAddress with a filled buffer address
write readCounter with the number of byte to send
Exit
UART TX is stopped because no more bytes to send are available
UART TX is running because bytes to send are available

Java Equivalent Code:

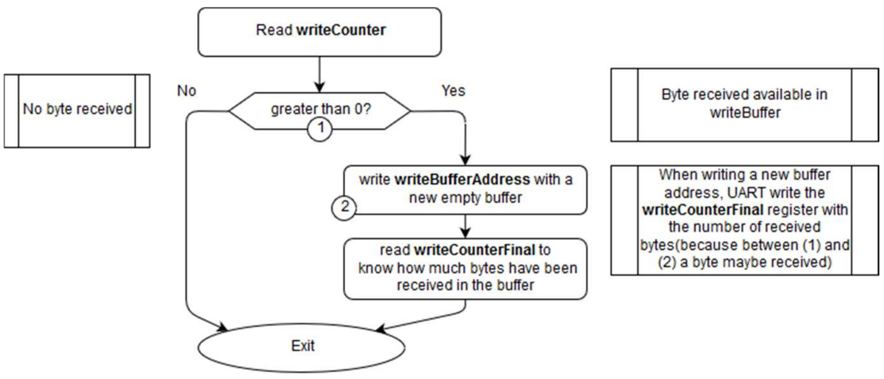
```
Val=CodESys.readSlave(READCOUNTER) ;
if(val>0) return -1; // for example -1 means "still busy"
CodeESys.writeSlave(READBUFFERADDRESS, newBuf);
CodeESys.writeSlave(READCOUNTER, nbByteToSend);
```

The MASTER bus only communicates with the SDRAM memory and never with the processor. The master hardware unit must have :

- a register which contains a read and/or a write buffer address.
- a register which contains the number of available bytes in the buffer

This part shows how to design a master unit with a simple example. We will consider we want to create an UART communication unit.

When there are some bytes to be sent on TX, the UART will use the DMA bus to read the memory at *readBufferAddress* and will send byte by byte a given number of bytes specified by the *readCounter* register. Each time a byte is sent, the *readBufferAddress* register is incremented and the *readCounter* register is decremented. When the *readCounter* register reaches 0 the UART stops to read. The reading will be restarted when the *readCounter* register will be set with a new value (strictly greater than 0). When the application needs to send new data, it must ensure the readCounter is equal to 0, set a new buffer (for example readbuffer2) in *readBufferAddress* and set the number of bytes to send in *readCounter*.


Read **writeCounter**
No byte received
No
greater than 0?
1
Yes
Byte received available in writeBuffer
write **writeBufferAddress** with a new empty buffer
2
read **writeCounterFinal** to know how much bytes have been received in the buffer
When writing a new buffer address, UART write the **writeCounterFinal** register with the number of received bytes(because between (1) and (2) a byte maybe received)
Exit

When a byte is received on RX, the UART uses the DMA bus to write the received byte in the SDRAM at the *writeBufferAddress*. The *writeBufferAddress* and the *writeCounter* will be incremented. When the application needs to read received data, it must read the *writeCounter*, check if it is greater than 0 and set a new empty buffer to replace the current one by setting the *writeBufferAddress*. When writing in *writeBufferAddress* register, the UART must also copy *writeCounter* in *writeCounterFinal* and reset *writeCounter* to 0. Next, the application must read *writeCounterFinal* to have the correct number of byte available in the buffer.

```
Java Equivalent Code:

Val=CodESys.readSlave(WRITECOUNTER) ;
if(val>0) return 0; // 0 means "no byte received"
CodeESys.writeSlave(WRITEBUFFERADDRESS, newBuf);
nbByteReceived=CodeESys.readSlave(WRITECOUNTERFIN
AL):
```

This mechanism of exchanging buffer is based on mechanism of *front* and *back* buffers. Front buffers are those who are used by the UART which are directly connected on RX and TX and the application works on back buffers.

The application uses the SLAVE API to read/write in the UART registers.

To simplify and accelerate the "exchange" (or swap) operation two methods have been created:

void swapReadBufMaster(int adRegBuf, byte newBuf[], int offsetBuf, int adRegCount, int availableByteInBuffer);

swapReadBufMaster will execute the following steps:

- Write (address of newBuf+offsetBuf) in the register at address adRegBuf.
- Write availableByteInBuffer in the register at address adRegCount

This method can be used like this:

```
Val=CodESys.readSlave(READCOUNTER) ;
if(val>0) return -1; // for example -1 means “still busy”
CodeESys.swapReadBufMaster(READBUFFERADDRESS, newBuf, 0, READCOUNTER, availableByteInNewBuf);
return Val;
```

int swapWriteBufMaster(int adRegBuf, byte newBuf[], int offsetBuf, int adRegCountFinal)

swapWriteBufMaster will execute the following steps:

- Write (address of newBuf + offsetBuf) int the register at address adRegBuf
- Read register adRegCountFinal
- Return value of register adRegCountFinal

This method can be used like this:

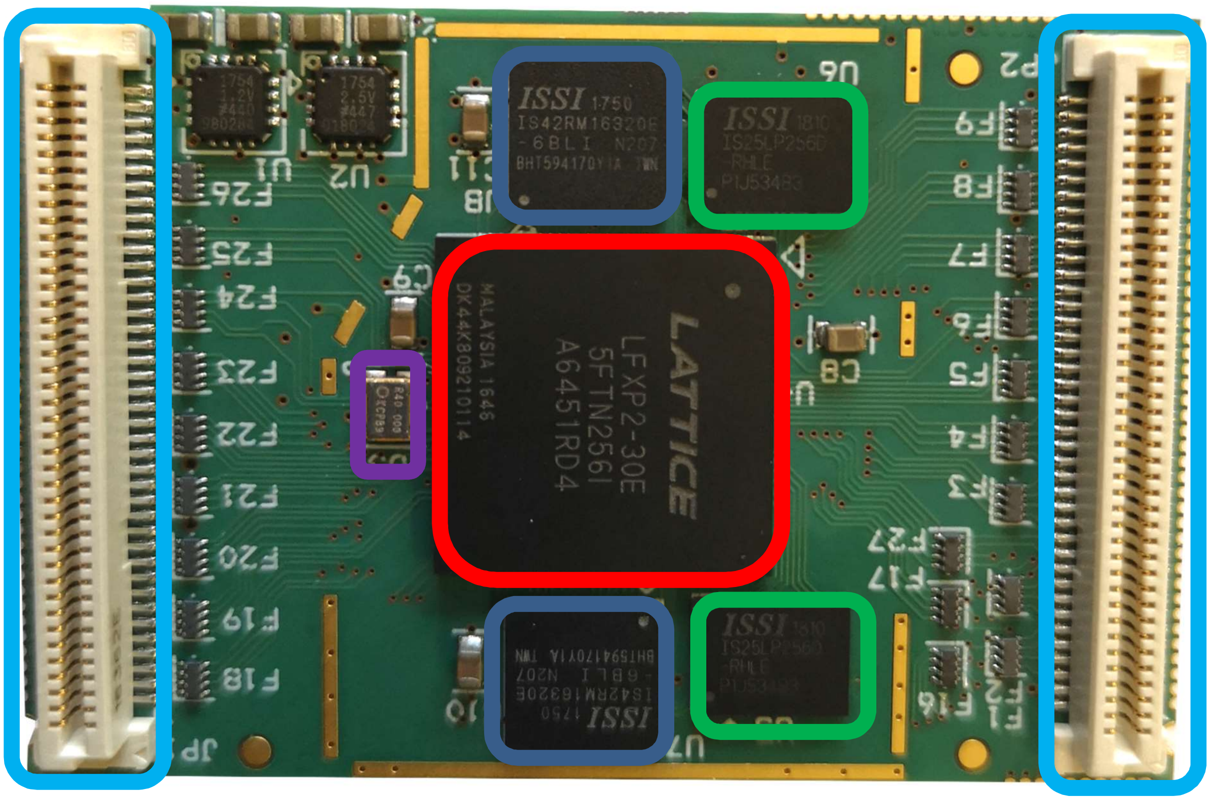
```
Val=CodESys.readSlave(WRITECOUNTER) ;
if(val==0) return 0; // for example -1 means “no available byte”
Val=CodeESys.swapWriteBufMaster(WRITEBUFFERADDRESS, newBuf, 0, WRITECOUNTERFINAL);
return Val;
```

6. CODESYS PLATFORM: ARKEON

CodESys is integrated in several SECAPEM products such as target acquisition systems or Secapem application specific display systems.

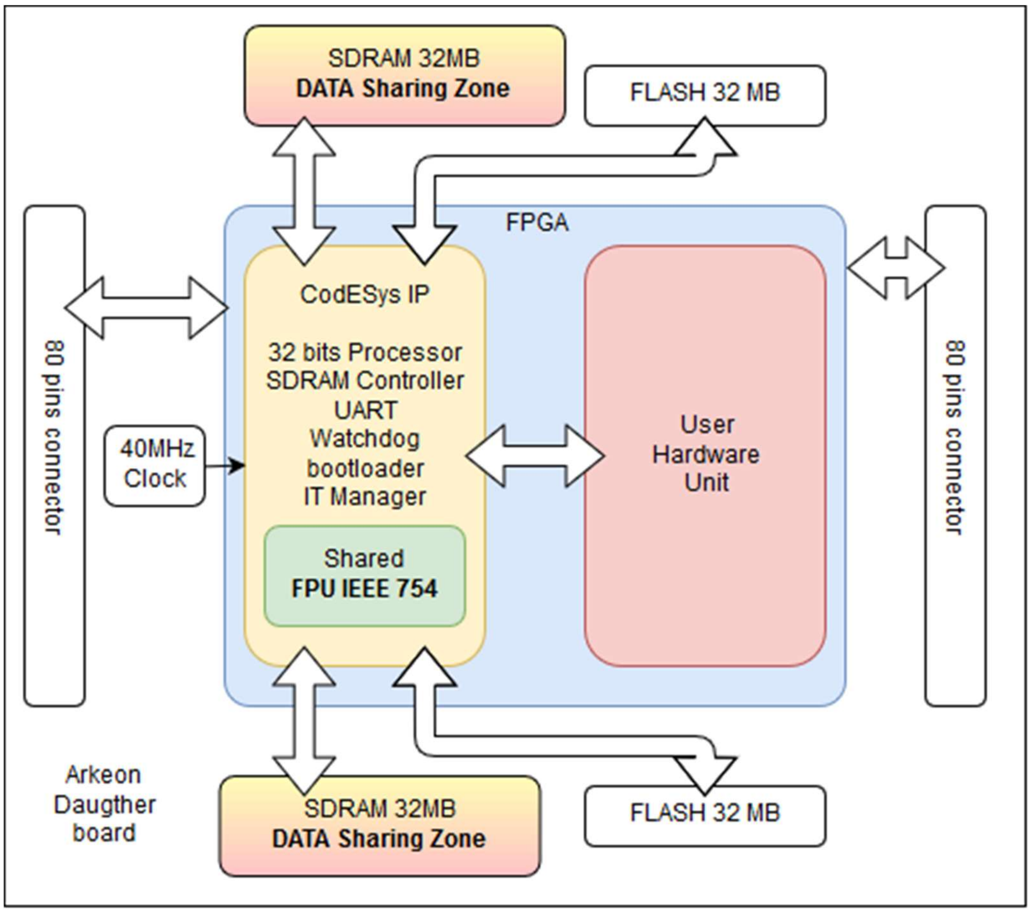
All systems are based on the same hardware platform, only the hardware unit and the application software are different. The hardware platform has been called ARKEON. It is a daughter board and it includes :

- 1 FPGA : Lattice LFXP2-30E (256 pins BGA package)
- 2 SDRAM of 32MB (total : 64 MB)
- 2 SPI Flash memory of 32 MB (total : 64 MB)
- 1 40MHz oscillator
- 2 80 pins connectors

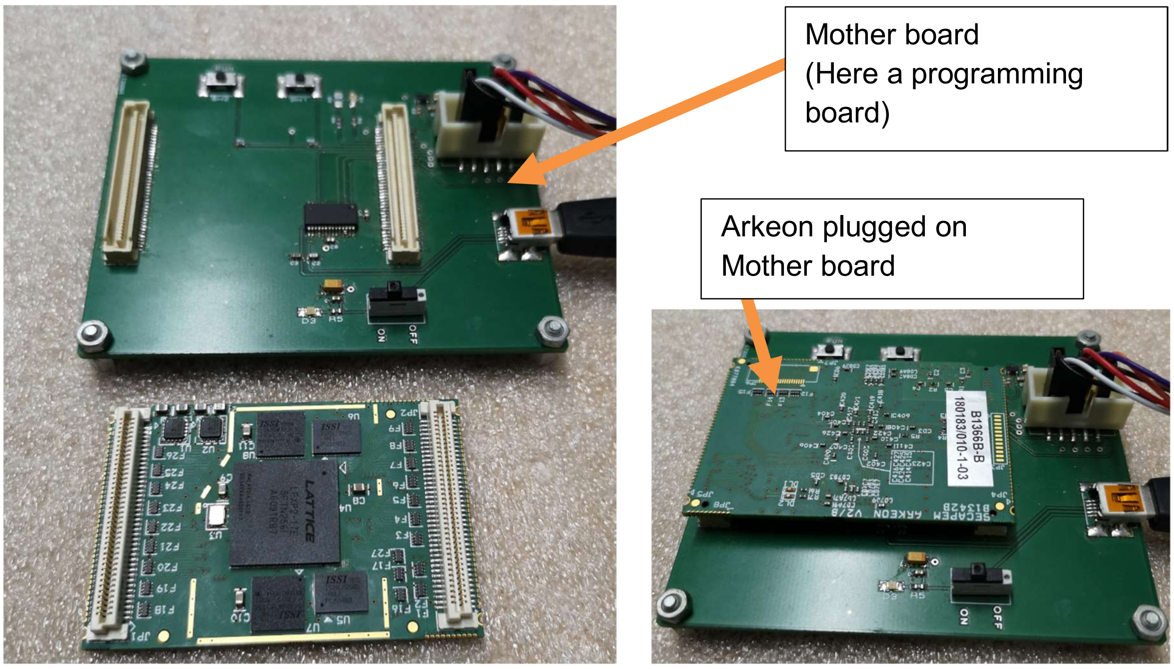


-

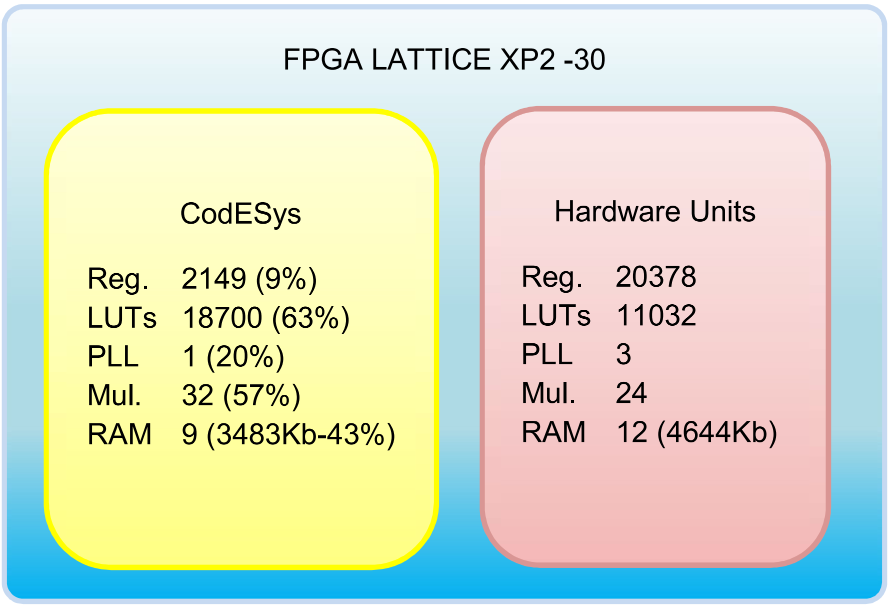
Lattice XP2 -30			
LUTs	EBR SRAM Blocks	EBR SRAM (Kbits)	Distributed RAM (Kbits)
29732	21	387	56
registers	18x18 multipliers	PLL+DLL	Configuration memory
22527	56	4+2	Internal Flash



Each connector pin is connected to the FPGA pins. The FPGA is configured following the mother board specification.



The CodESys IP (including processor, FPU, SDRAM controller) uses 63% of the available LUTs in Lattice XP2 -30. The figure below shows the occupation of the CodESys ip in the Lattice XP2 -30 FPGA.



FPGA LATTICE XP2 -30

CodESys		Hardware Units	
Reg.	2149 (9%)	Reg.	20378
LUTs	18700 (63%)	LUTs	11032
PLL	1 (20%)	PLL	3
Mul.	32 (57%)	Mul.	24
RAM	9 (3483Kb-43%)	RAM	12 (4644Kb)

If more space is needed, SECAPEM can design boards with some bigger FPGAs (Lattice, Xilinx, Altera, Microsemi (Microchip)).

Diamond is the tool to develop hardware IP for Lattice FPGAs (either in VHDL or Verilog).

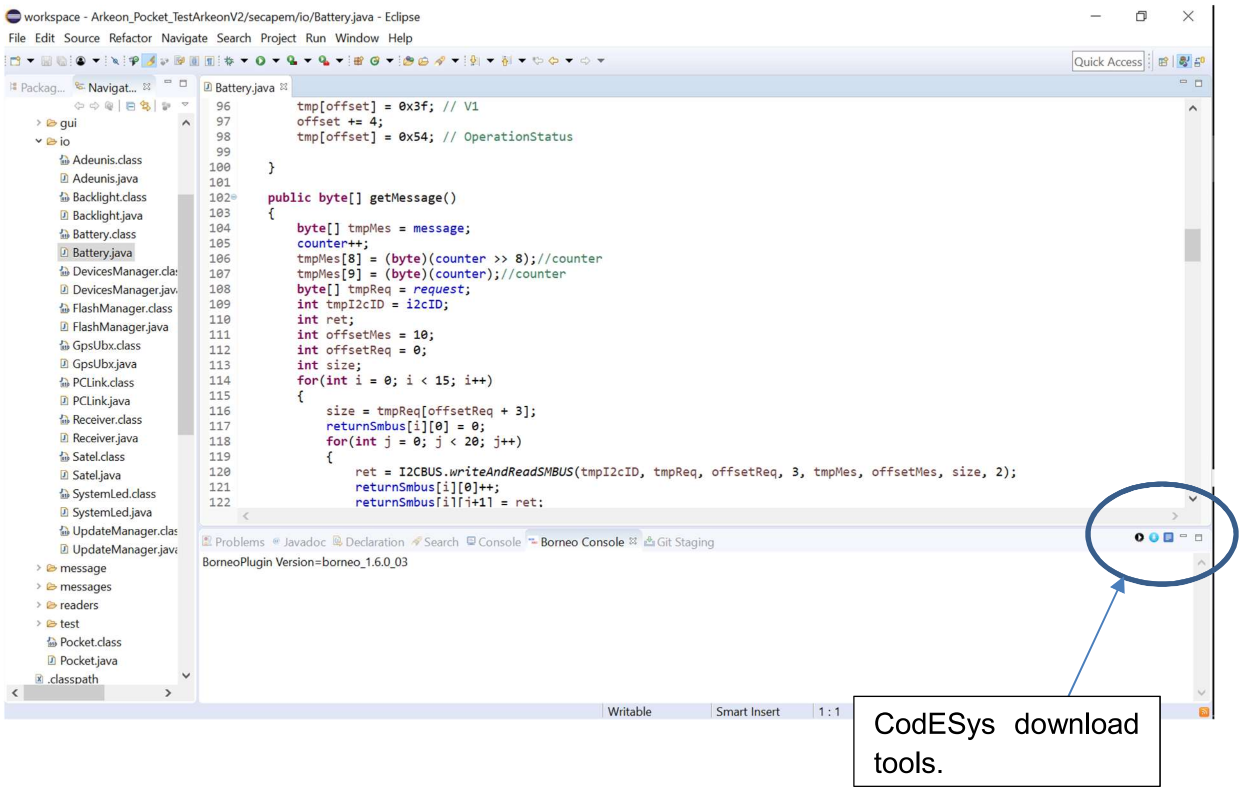
Downloadable at:

http://www.latticesemi.com/en/Products/DesignSoftwareAndIP/FPGAandLDS/LatticeDiamond

LATTICE
DIAMOND
DESIGN SOFTWARE

7. CODESYS: SOFTWARE DEVELOPMENT TOOL

To develop application CodESys uses the Java language. A plugin for Eclipse has been developed for CodESys.



```
        tmp[offset] = 0x3f; // V1
        offset += 4;
        tmp[offset] = 0x54; // OperationStatus

    }

    public byte[] getMessage()
    {
        byte[] tmpMes = message;
        counter++;
        tmpMes[8] = (byte)(counter >> 8);//counter
        tmpMes[9] = (byte)(counter);//counter
        byte[] tmpReq = request;
        int tmpI2cID = i2cID;
        int ret;
        int offsetMes = 10;
        int offsetReq = 0;
        int size;
        for(int i = 0; i < 15; i++)
        {
            size = tmpReq[offsetReq + 3];
            returnSmbus[i][0] = 0;
            for(int j = 0; j < 20; j++)
            {
                ret = I2CBUS.writeAndReadSMBUS(tmpI2cID, tmpReq, offsetReq, 3, tmpMes, offsetMes, size, 2);
                returnSmbus[i][0]++;
                returnSmbus[i][j+1] = ret;
```

```
BorneoPlugin Version=borneo_1.6.0_03
```