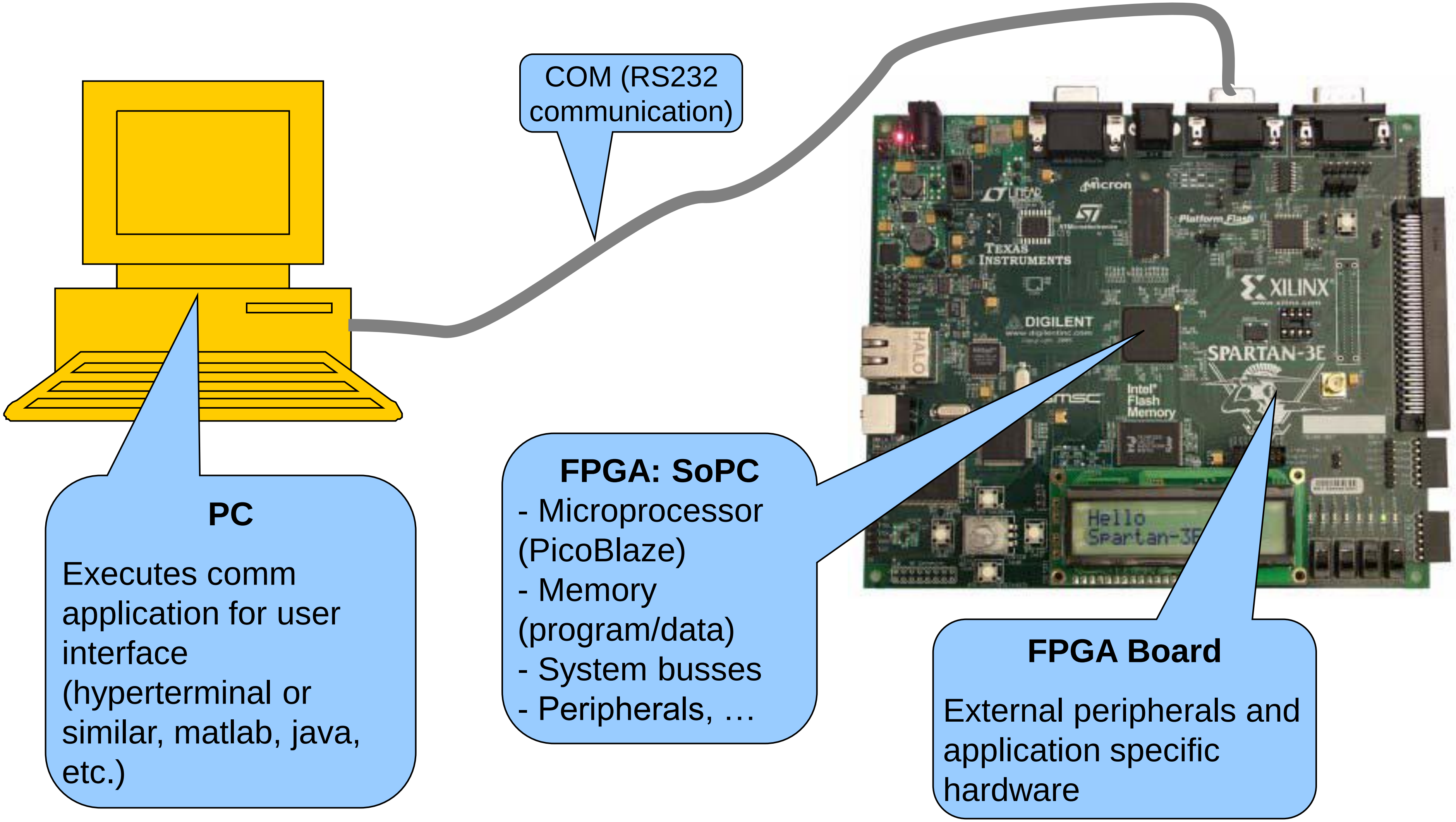
SoC design with PicoBlaze IP-core

On DS3Estarter Platform Board

Objective: designing a simple SoPC



COM (RS232 communication)

PC

Executes comm application for user interface (hyperterminal or similar, matlab, java, etc.)

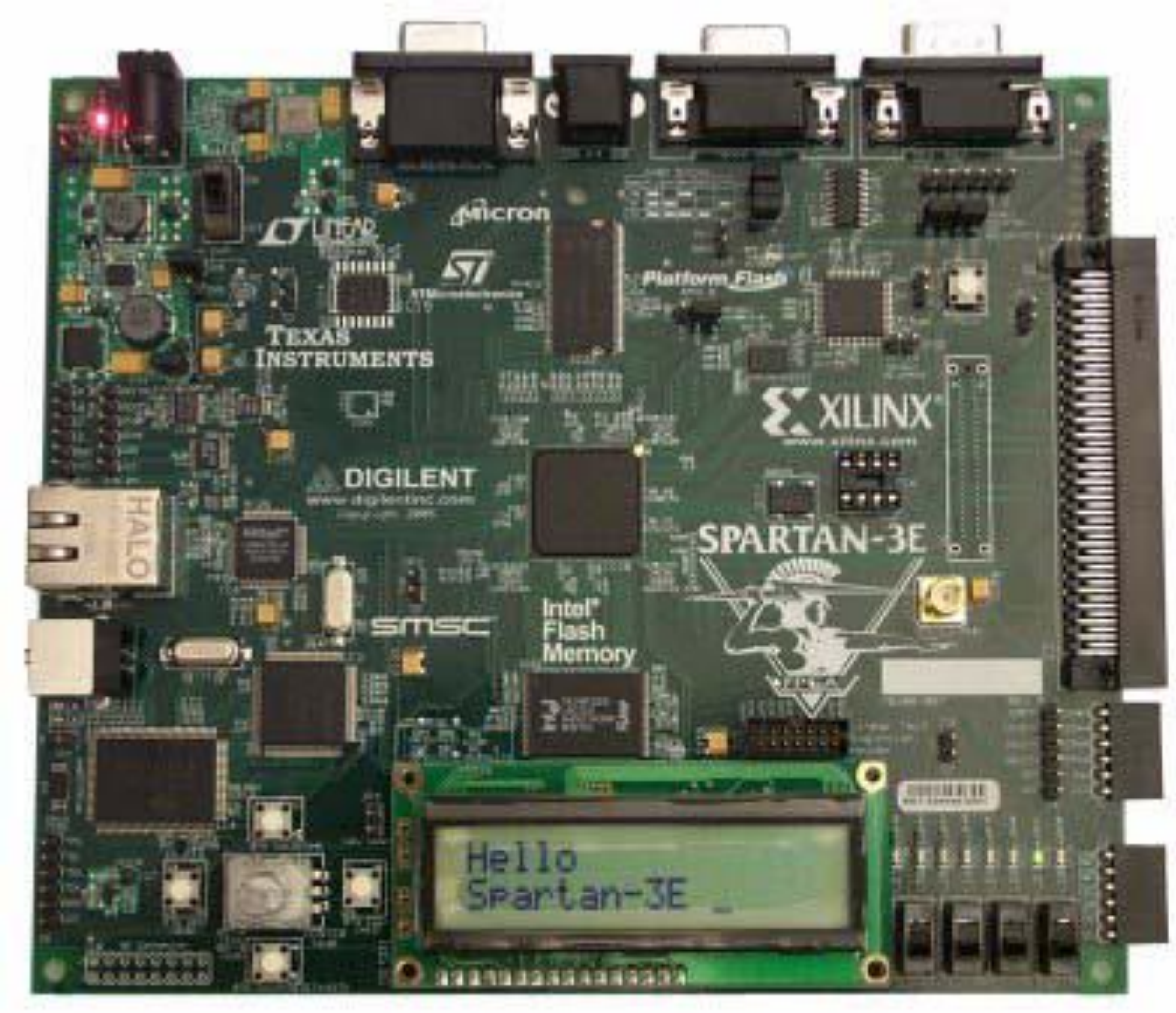
FPGA: SoPC

- Microprocessor (PicoBlaze)
- Memory (program/data)
- System busses
- Peripherals, …

FPGA Board

External peripherals and application specific hardware

Lab Board: Digilent S3E Starter Kit*



*check complete datasheet in aulavirtual for detailed info

Lab Board: Digilent S3E Starter Kit

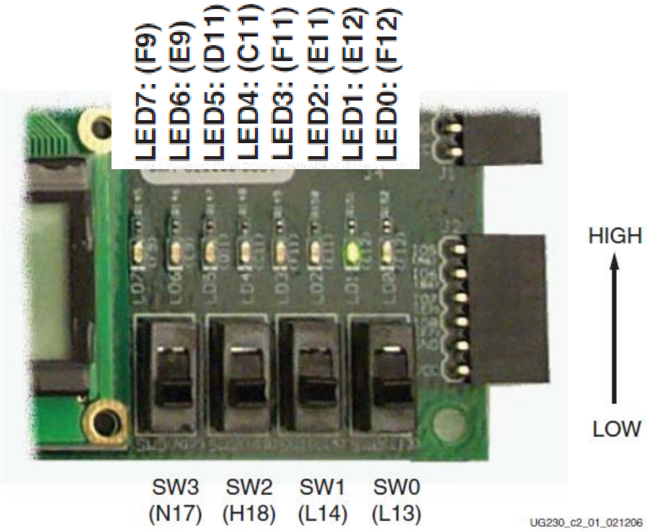
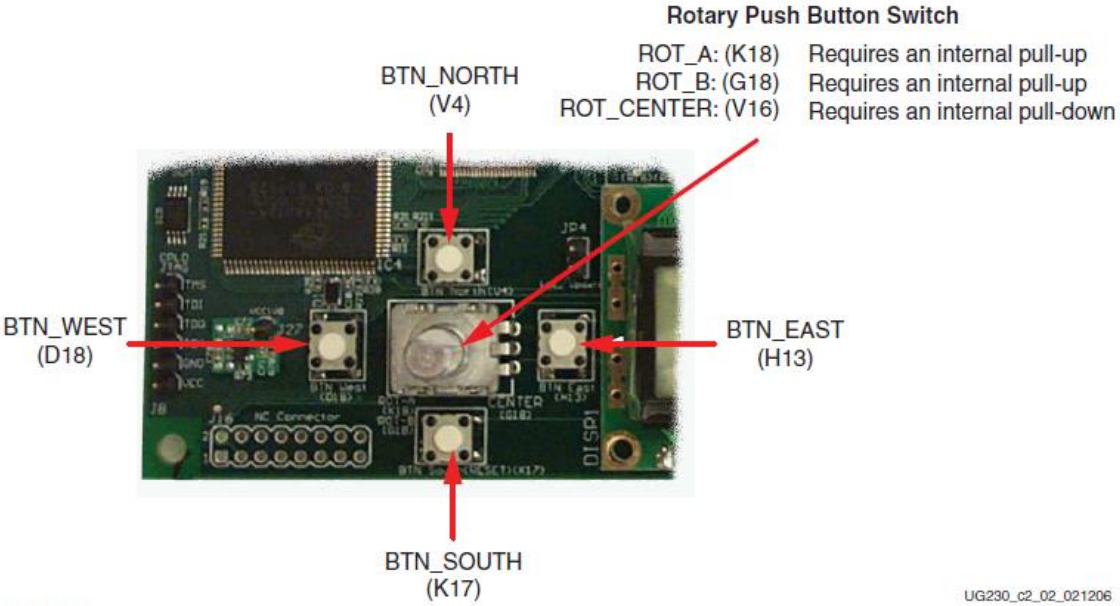


Figure 2-1: **Four Slide Switches**

When in the UP or ON position, a switch connects the FPGA pin to 3.3V, a logic High.

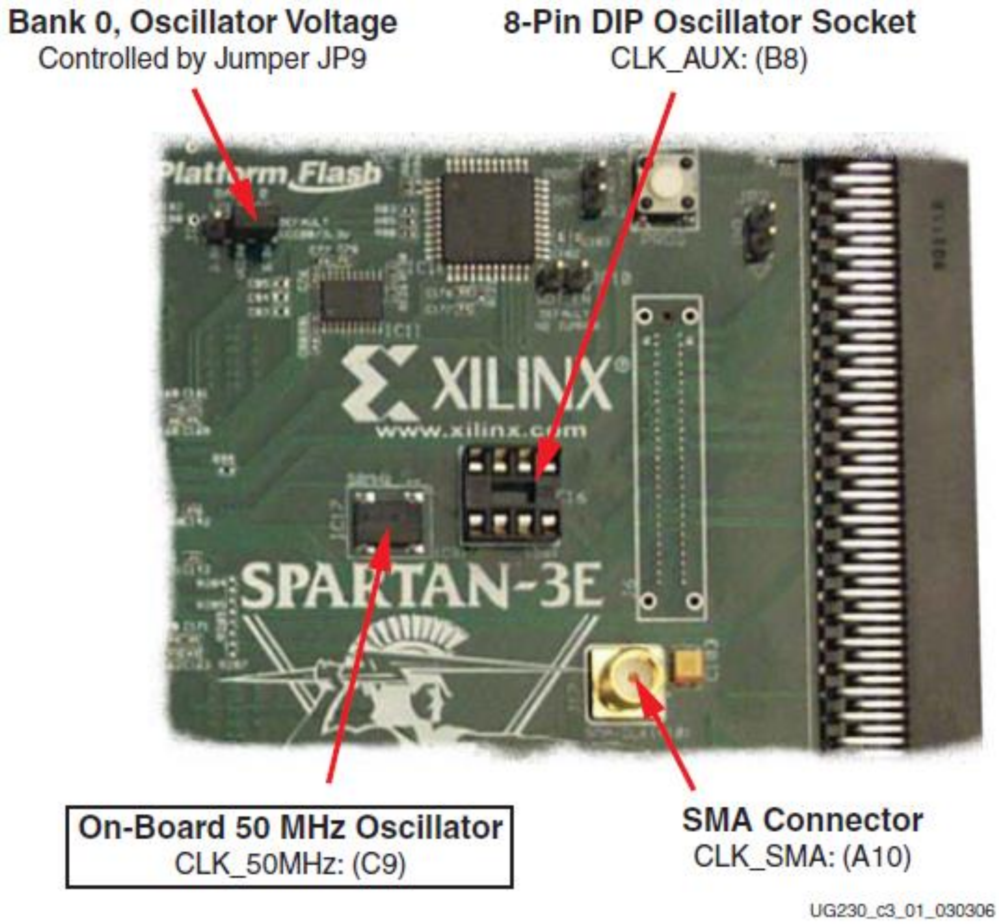

Rotary Push Button Switch
ROT_A: (K18) Requires an internal pull-up
ROT_B: (G18) Requires an internal pull-up
ROT_CENTER: (V16) Requires an internal pull-down
BTN_NORTH (V4)
BTN_WEST (D18)
BTN_EAST (H13)
BTN_SOUTH (K17)
UG230_c2_02_021206

Notes:

1. All BTN_* push-button inputs require an internal pull-down resistor.
2. BTN_SOUTH is also used as a soft reset in some FPGA applications.

Figure 2-3: **Four Push-Button Switches Surround Rotary Push-Button Switch**

Lab Board: Digilent S3E Starter Kit



Bank 0, Oscillator Voltage
Controlled by Jumper JP9

8-Pin DIP Oscillator Socket
CLK_AUX: (B8)

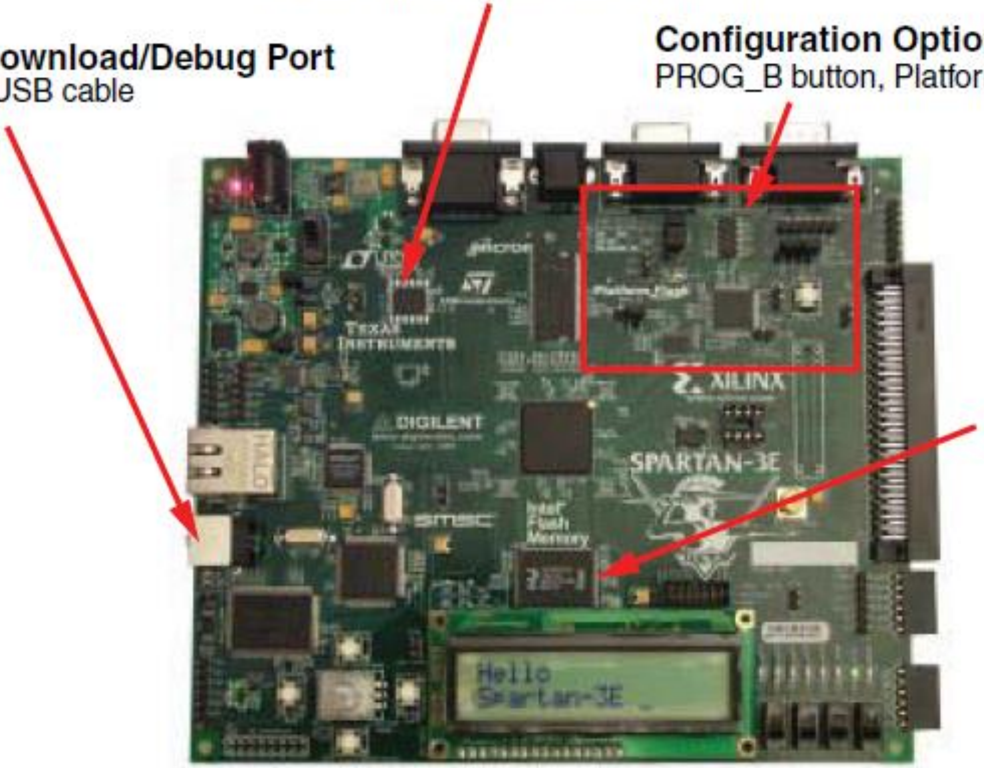
On-Board 50 MHz Oscillator
CLK_50MHz: (C9)

SMA Connector
CLK_SMA: (A10)

UG230_c3_01_030306

Figure 3-1: **Available Clock Inputs**

Lab Board: Digilent S3E Starter Kit



16 Mbit ST Micro SPI Serial Flash
Serial Peripheral Interface (SPI) mode

USB-based Download/Debug Port
Uses standard USB cable

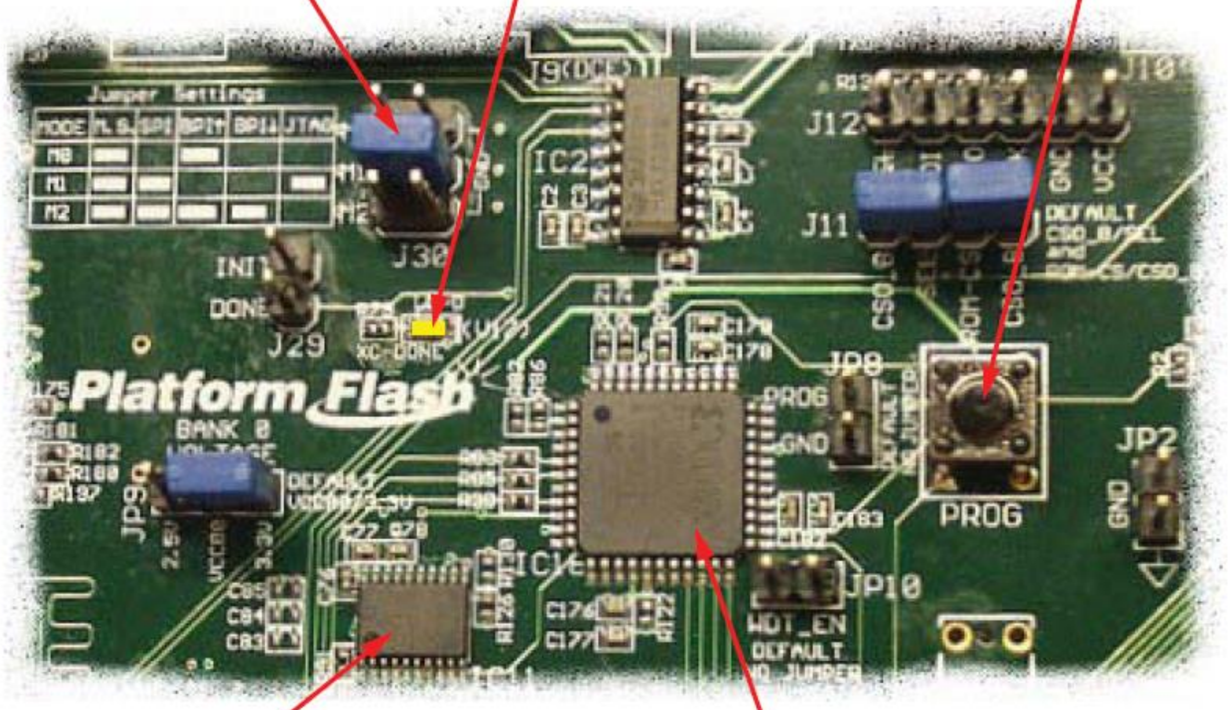
Configuration Options
PROG_B button, Platform Flash PROM, mode pins

128 Mbit Intel StrataFlash
Parallel NOR Flash memory
Byte Peripheral Interface (BPI) mode

UG230_c4_01_022006

Figure 4-1: **Spartan-3E Starter Kit FPGA Configuration Options**

Lab Board: Digilent S3E Starter Kit



Configuration Mode Jumper Settings (Header J30)
Select between three on-board configuration sources

DONE Pin LED
Lights up when FPGA successfully configured

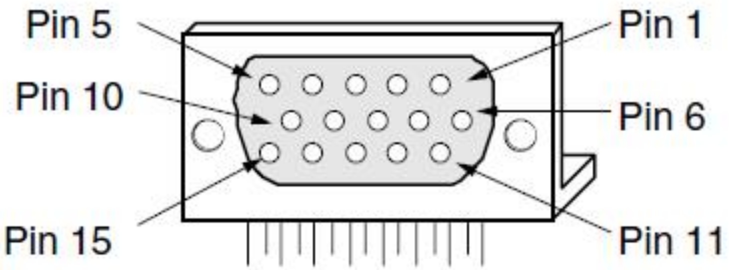
PROG_B Push Button Switch
Press and release to restart configuration

4 Mbit Xilinx Platform Flash PROM
Configuration storage for Master Serial mode

64 Macrocell Xilinx XC2C64A CoolRunner CPLD
Controller upper address lines in BPI mode and
Platform Flash chip select (User programmable)

UG230_c4_02_030906

Lab Board: Digilent S3E Starter Kit



DB15 VGA Connector
(front view)

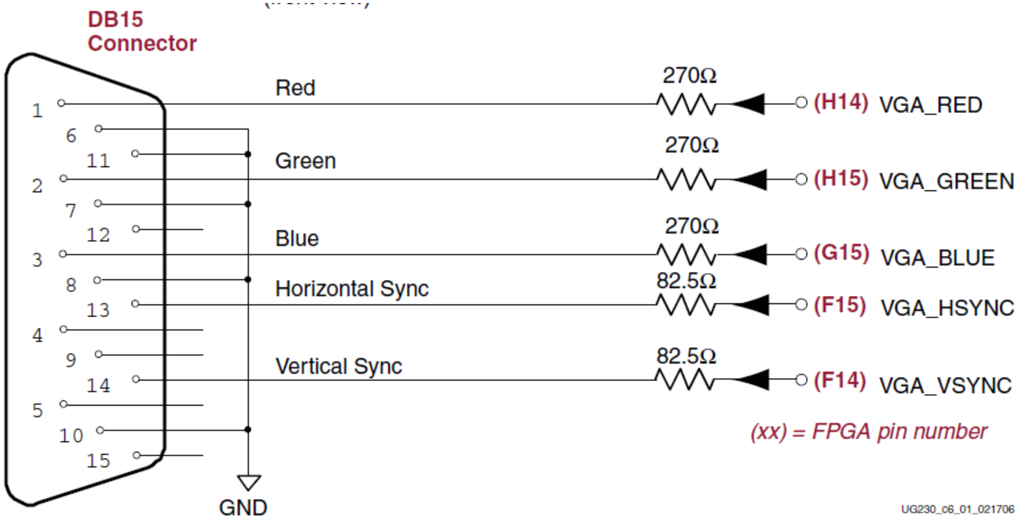
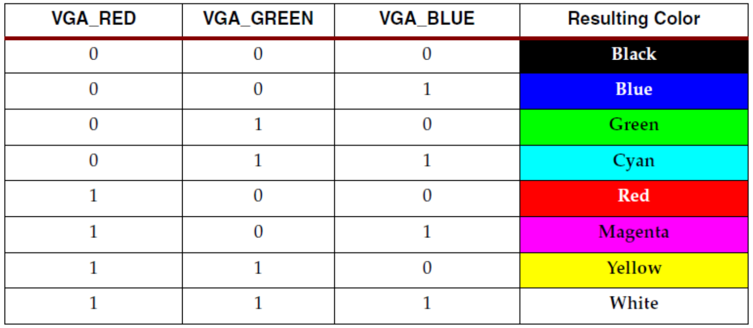
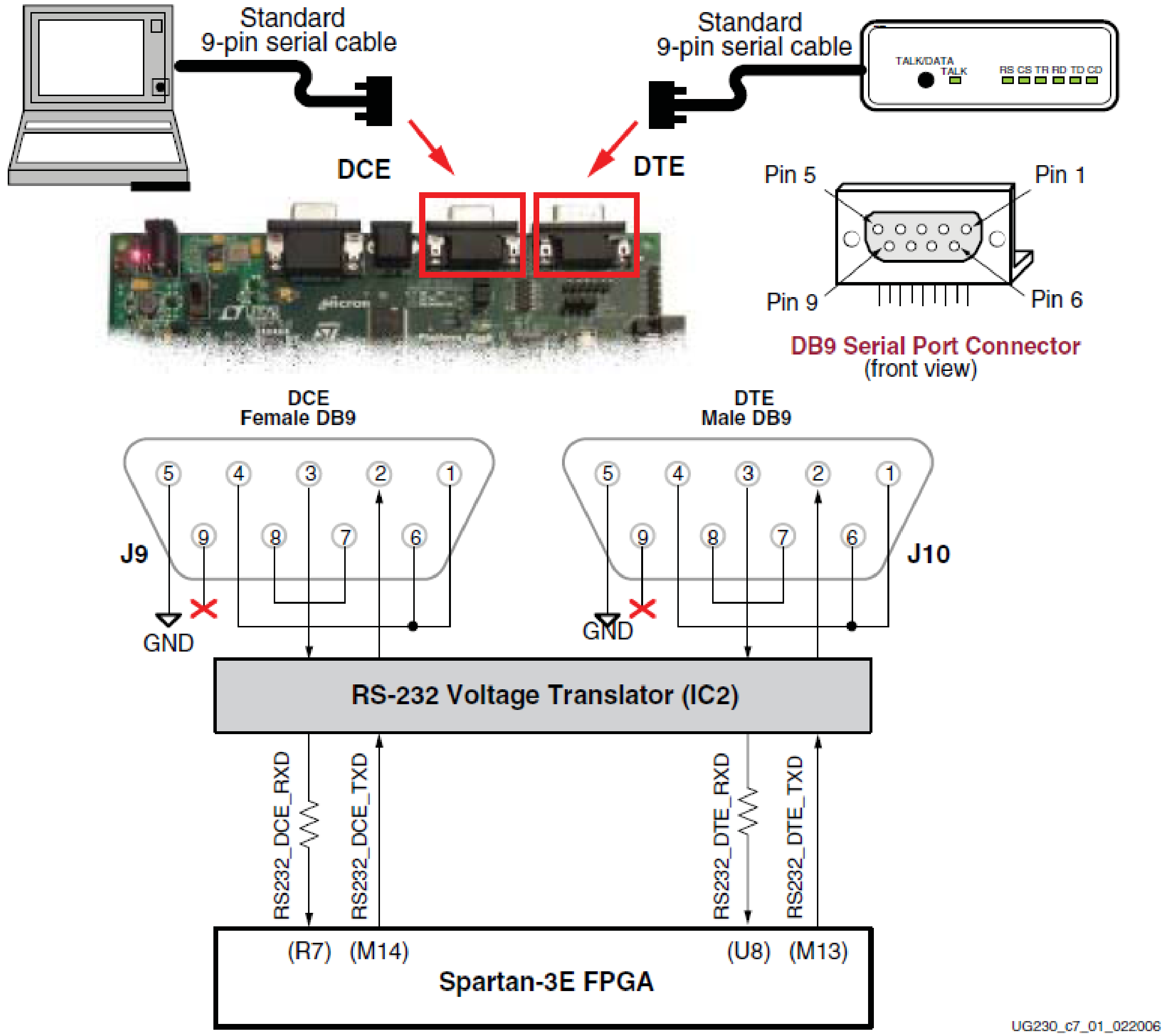


Table 6-1: **3-Bit Display Color Codes**

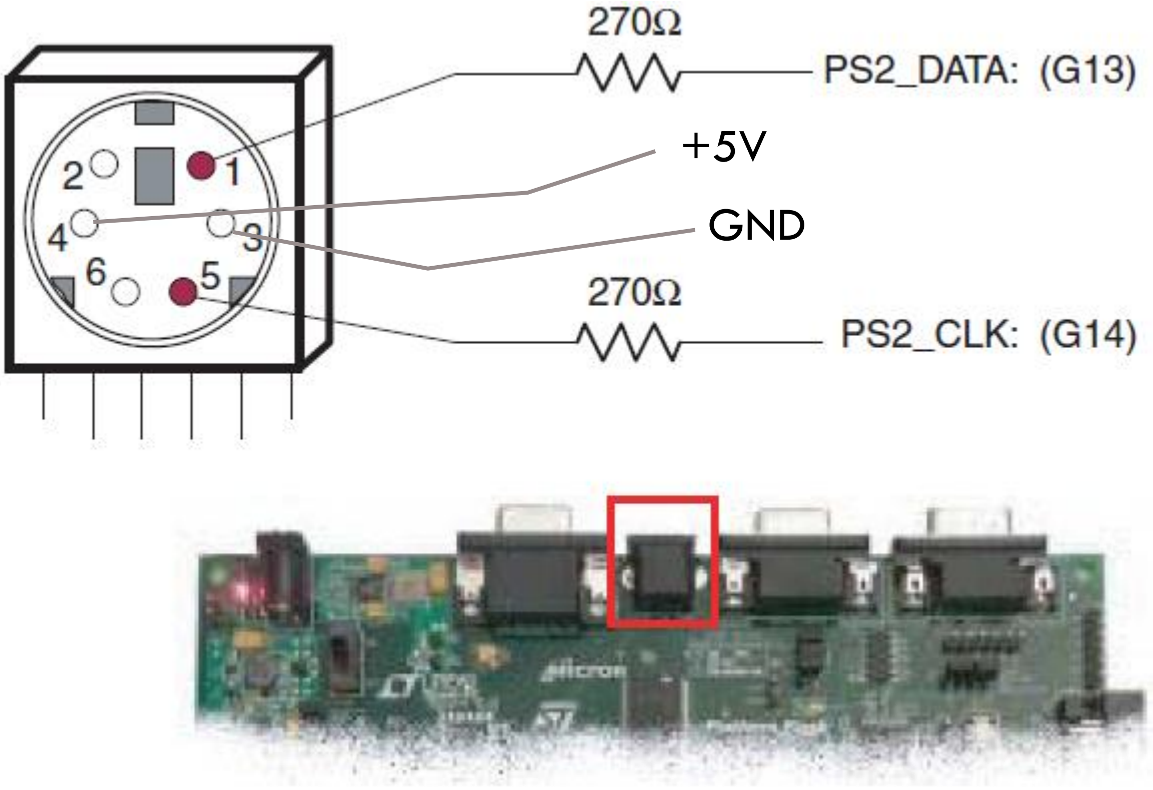


VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Lab Board: Digilent S3E Starter Kit

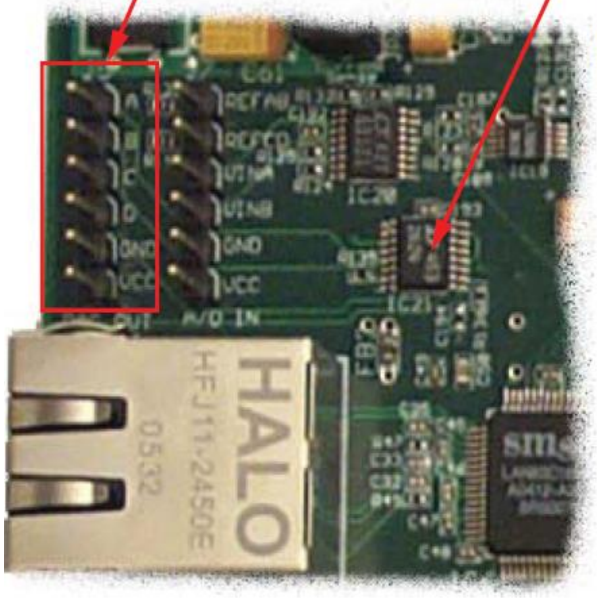


Lab Board: Digilent S3E Starter Kit



UG230_c8_01_021806

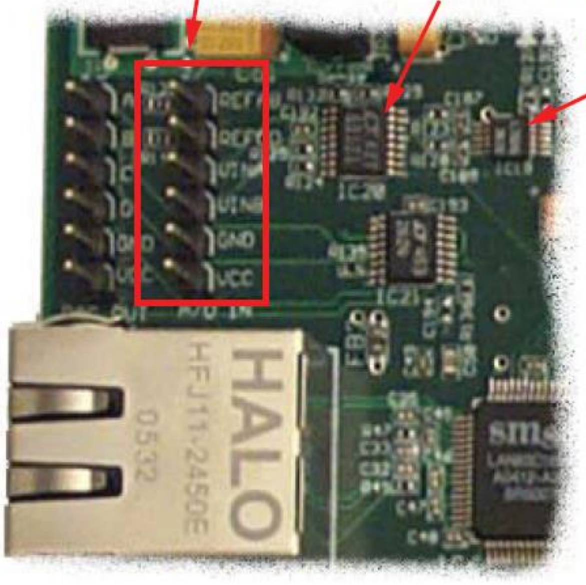
Lab Board: Digilent S3E Starter Kit



6-pin DAC Header (J5)

Linear Tech LTC2624 Quad DAC

SPI_MOSI: (T4)
SPI_MISO: (N10)
SPI_SCK: (U16)
DAC_CS: (N8)
DAC_CLR: (P8)



6-pin ADC Header (J7)

Linear Tech LTC1407A-1 Dual A/D

SPI_SCK: (U16)
AD_CONV: (P11)
SPI_MISO: (N10)

Linear Tech LTC6912-1 Dual Amp

SPI_MOSI: (T4)
AMP_CS: (N7)
SPI_SCK: (U16)
AMP_SHDN: (P7)
AMP_DOUT: (E18)

Lab Board: Digilent S3E Starter Kit

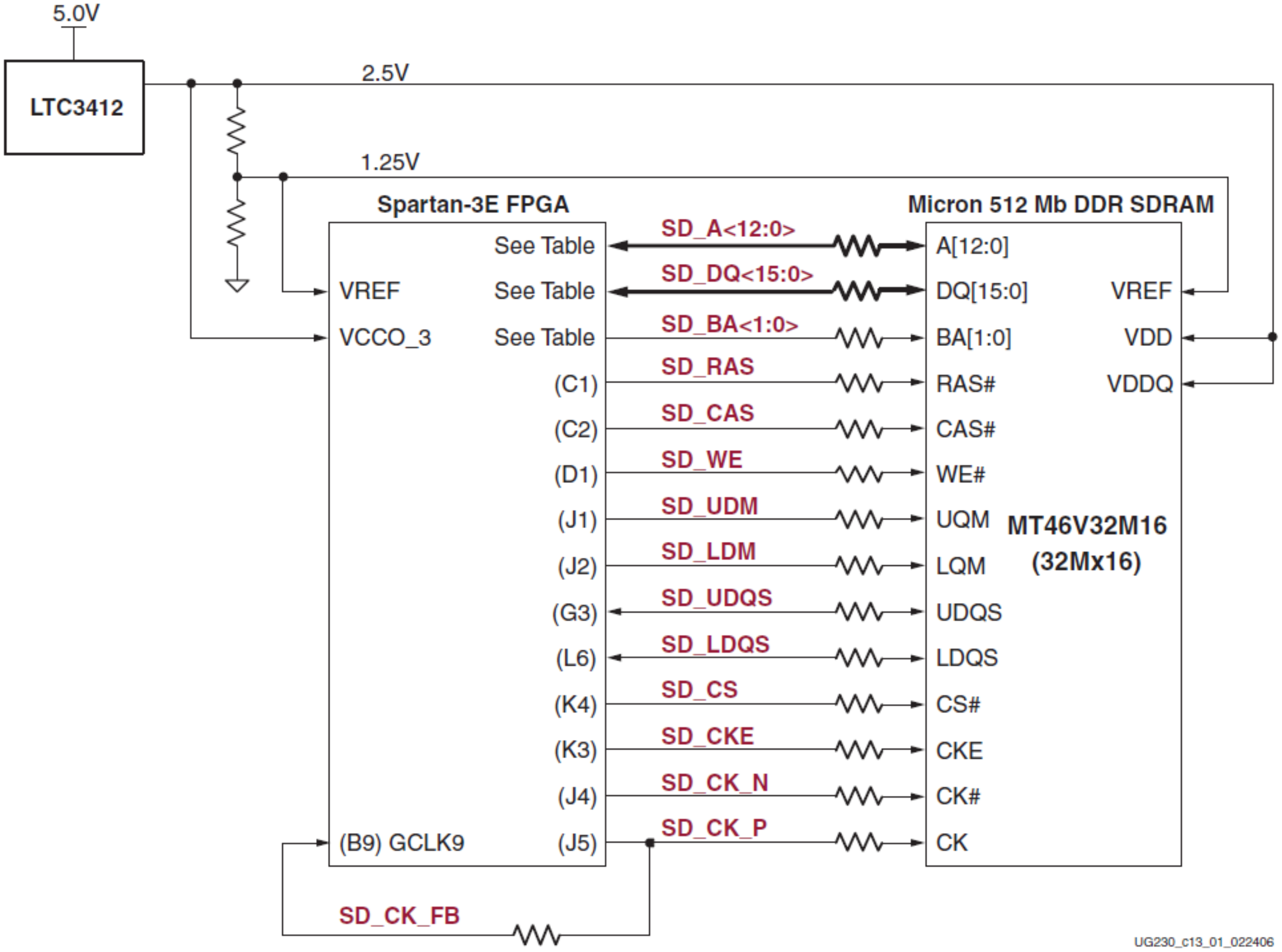
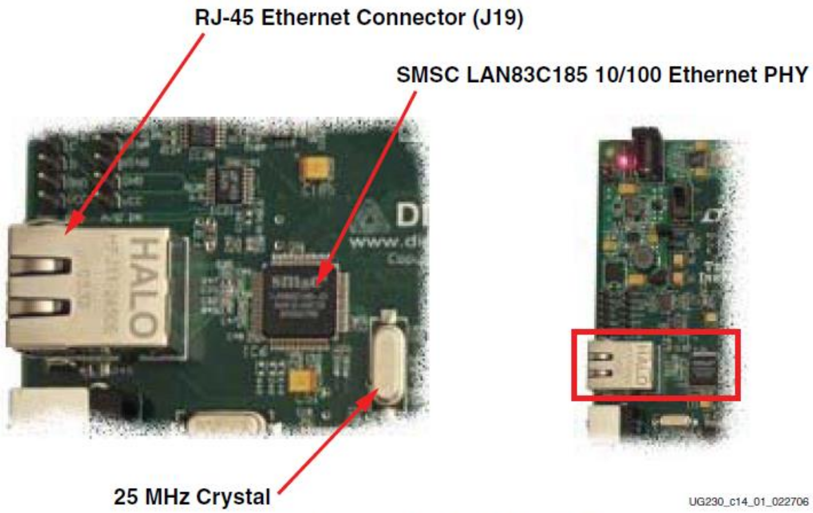


Figure 13-1: **FPGA Interface to Micron 512 Mbit DDR SDRAM**

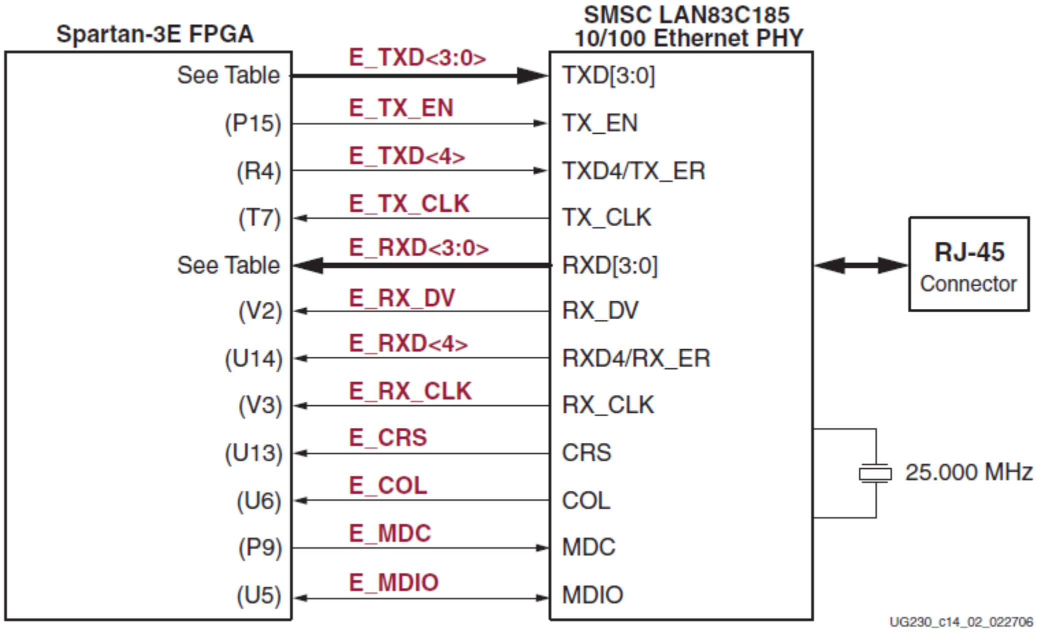
Lab Board: Digilent S3E Starter Kit



RJ-45 Ethernet Connector (J19)
SMSC LAN83C185 10/100 Ethernet PHY
25 MHz Crystal

UG230_c14_01_022706

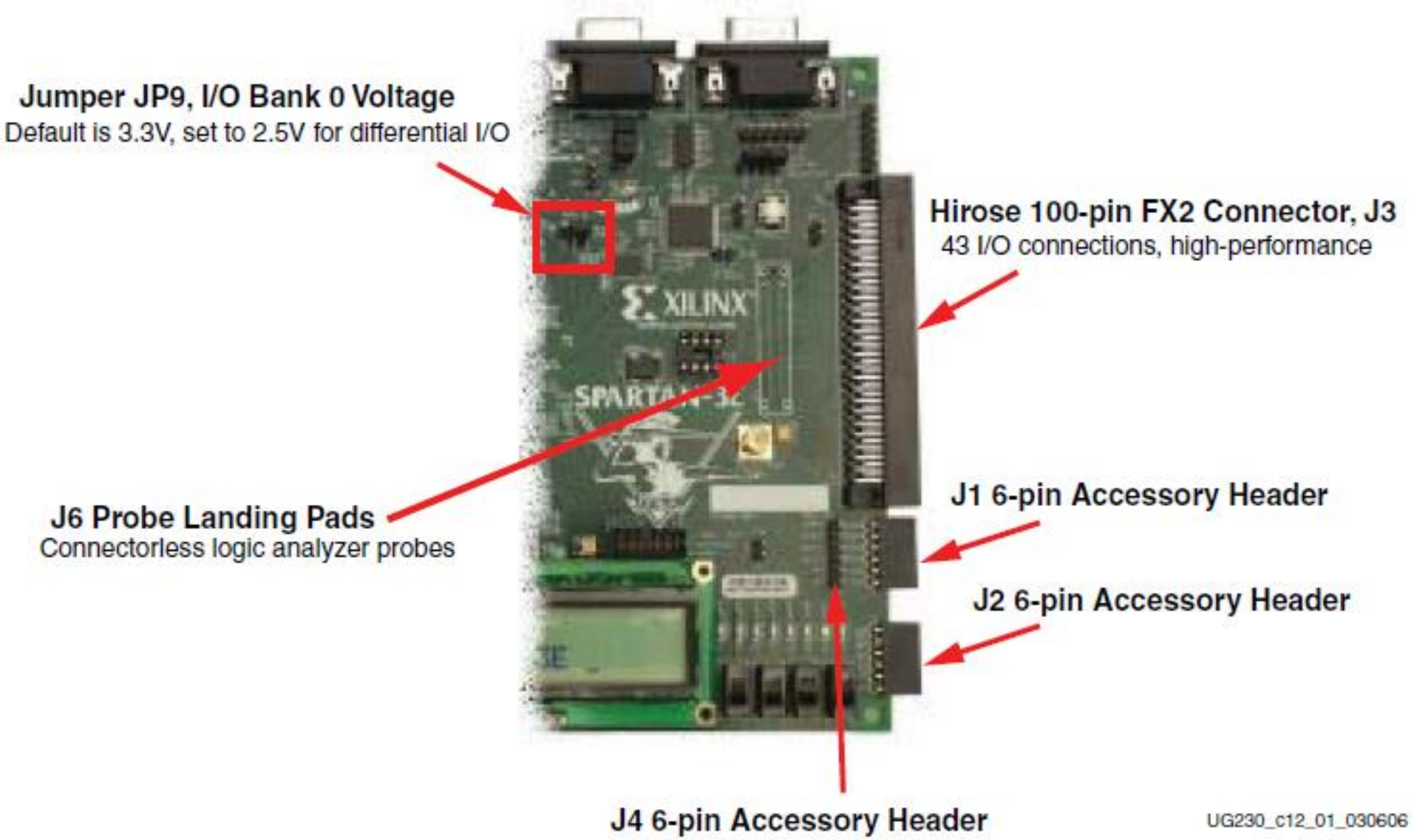
Figure 14-1: **10/100 Ethernet PHY with RJ-45 Connector**



UG230_c14_02_022706

Figure 14-2: **FPGA Connects to Ethernet PHY via MII**

Lab Board: Digilent S3E Starter Kit



Jumper JP9, I/O Bank 0 Voltage
Default is 3.3V, set to 2.5V for differential I/O

Hirose 100-pin FX2 Connector, J3
43 I/O connections, high-performance

J6 Probe Landing Pads
Connectorless logic analyzer probes

J1 6-pin Accessory Header

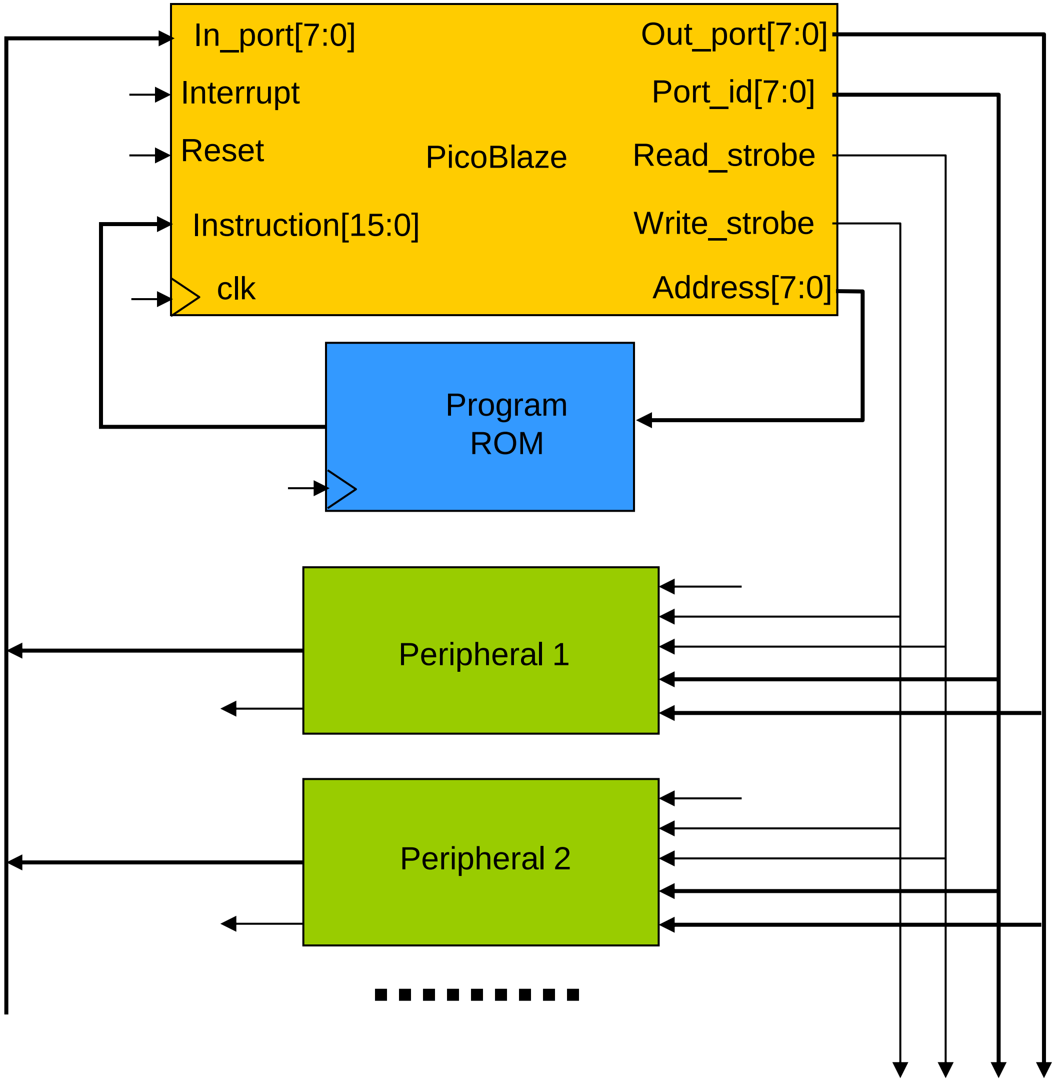
J2 6-pin Accessory Header

J4 6-pin Accessory Header

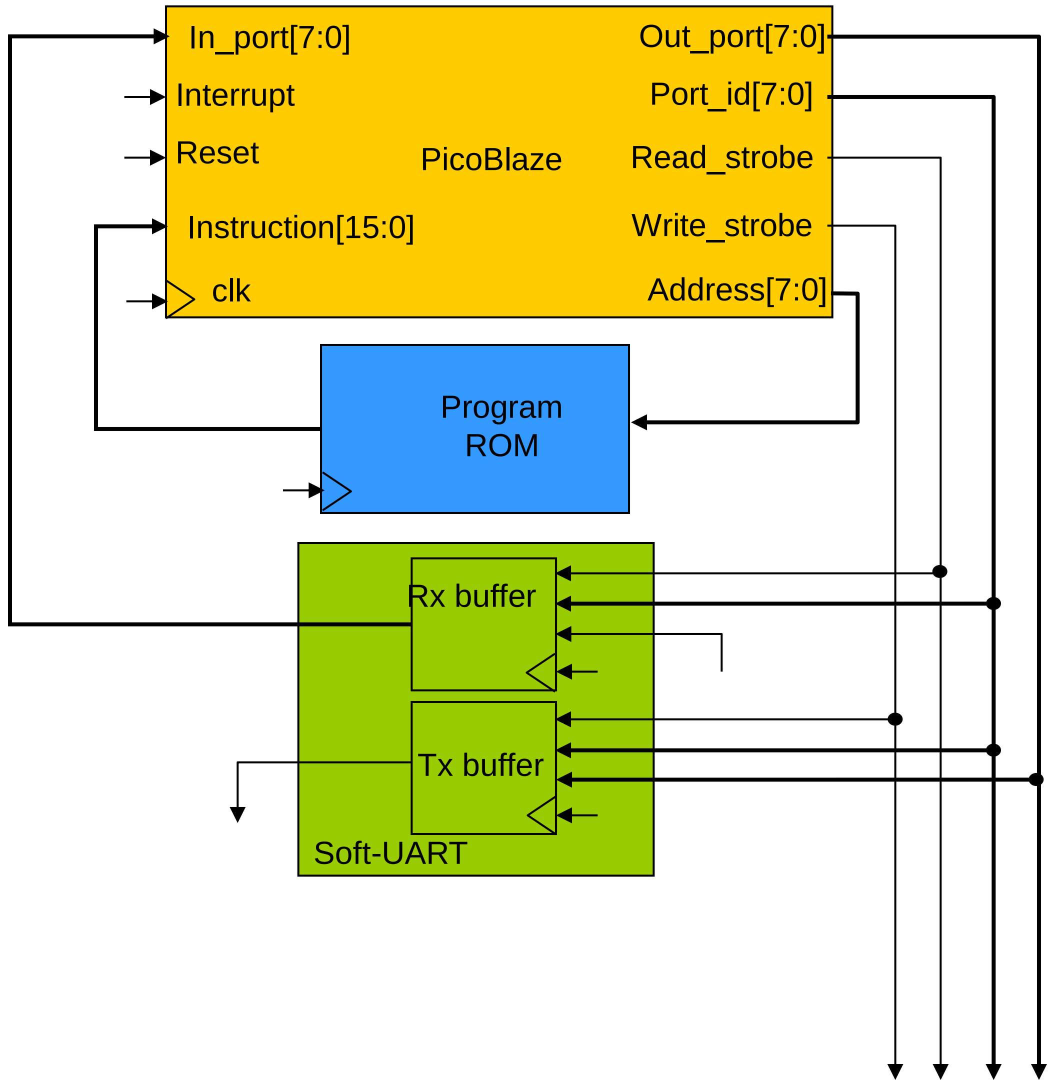
UG230_c12_01_030606

Figure 15-1: **Expansion Headers**

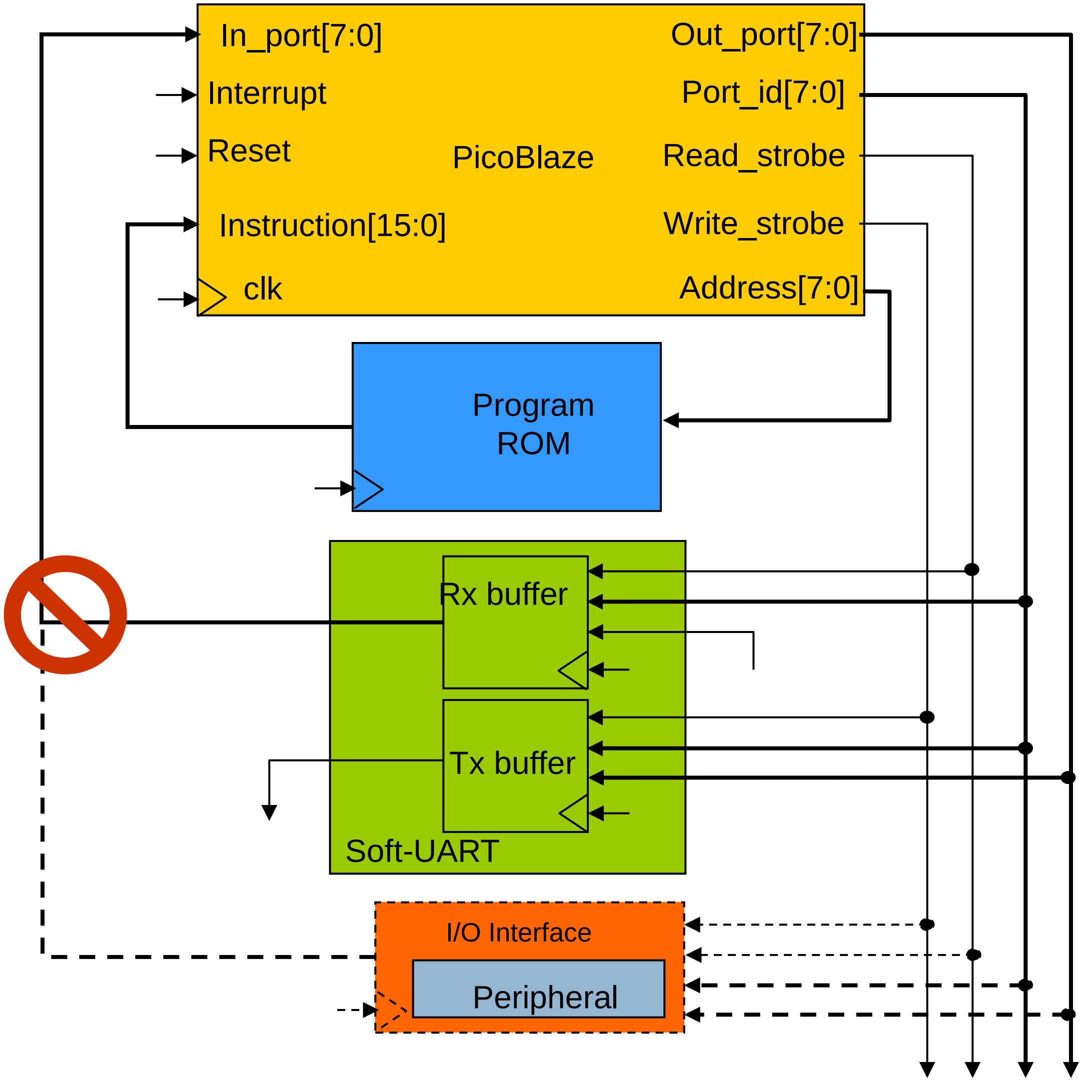
Architecture of a PicoBlaze based SoPC



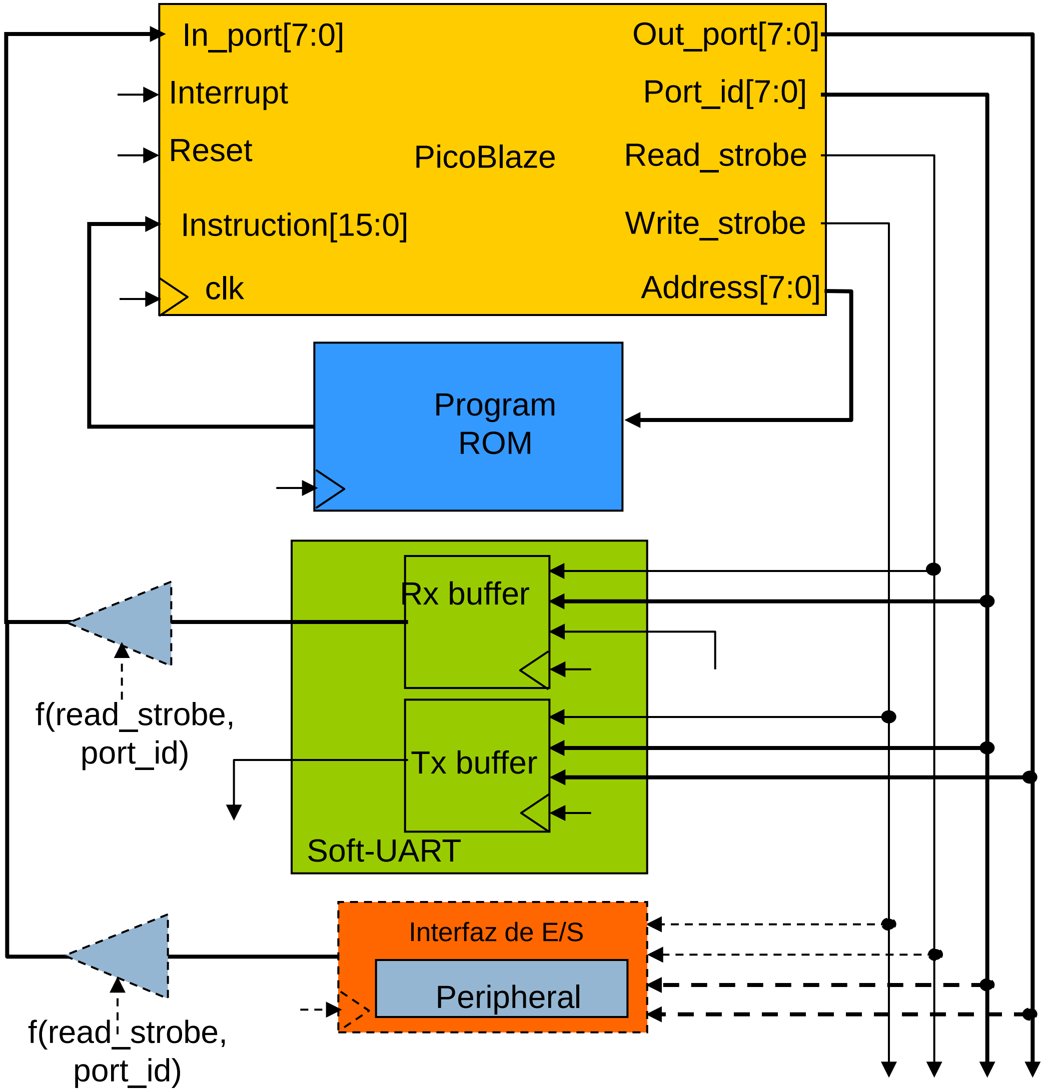
Example: simplest system ("hello world" lab)



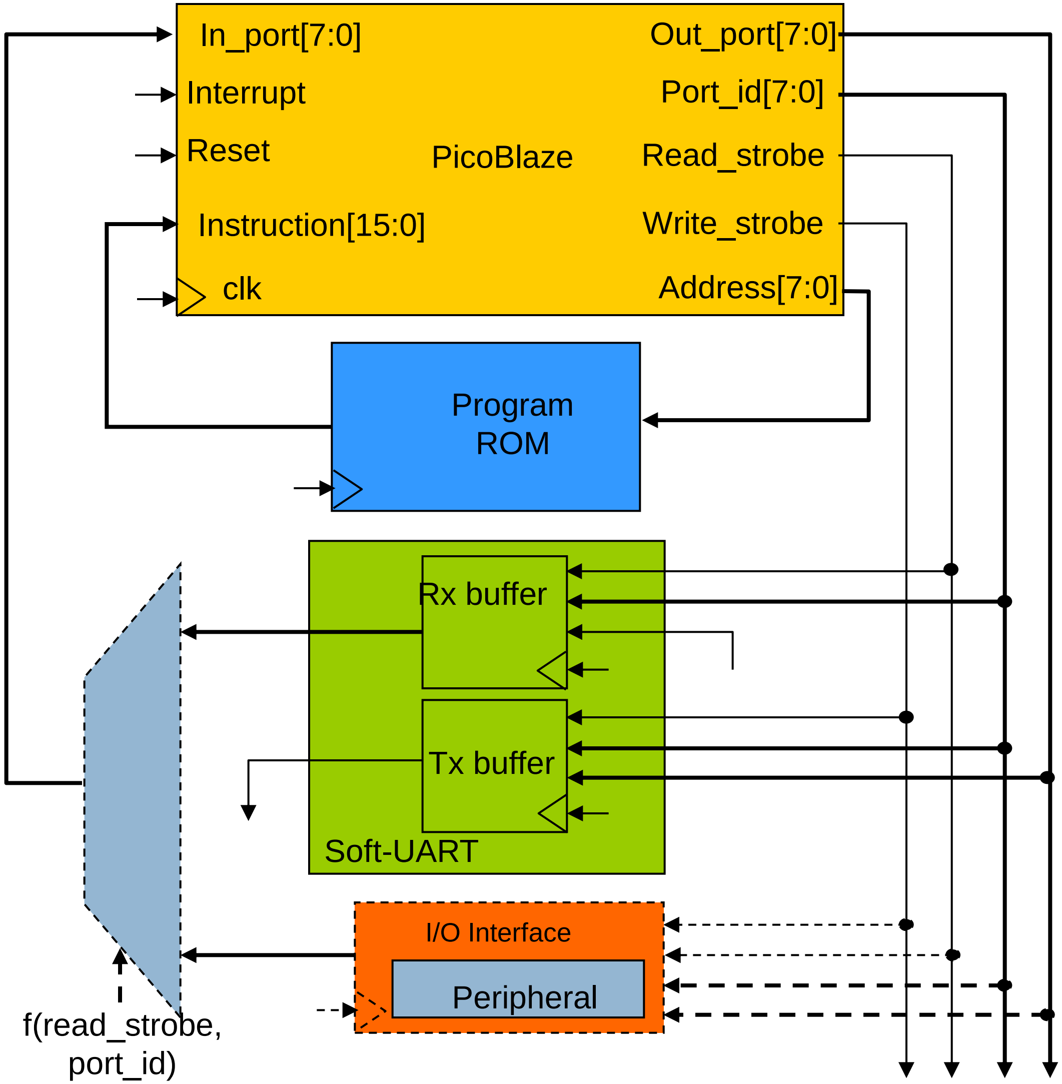
Adding more peripherals



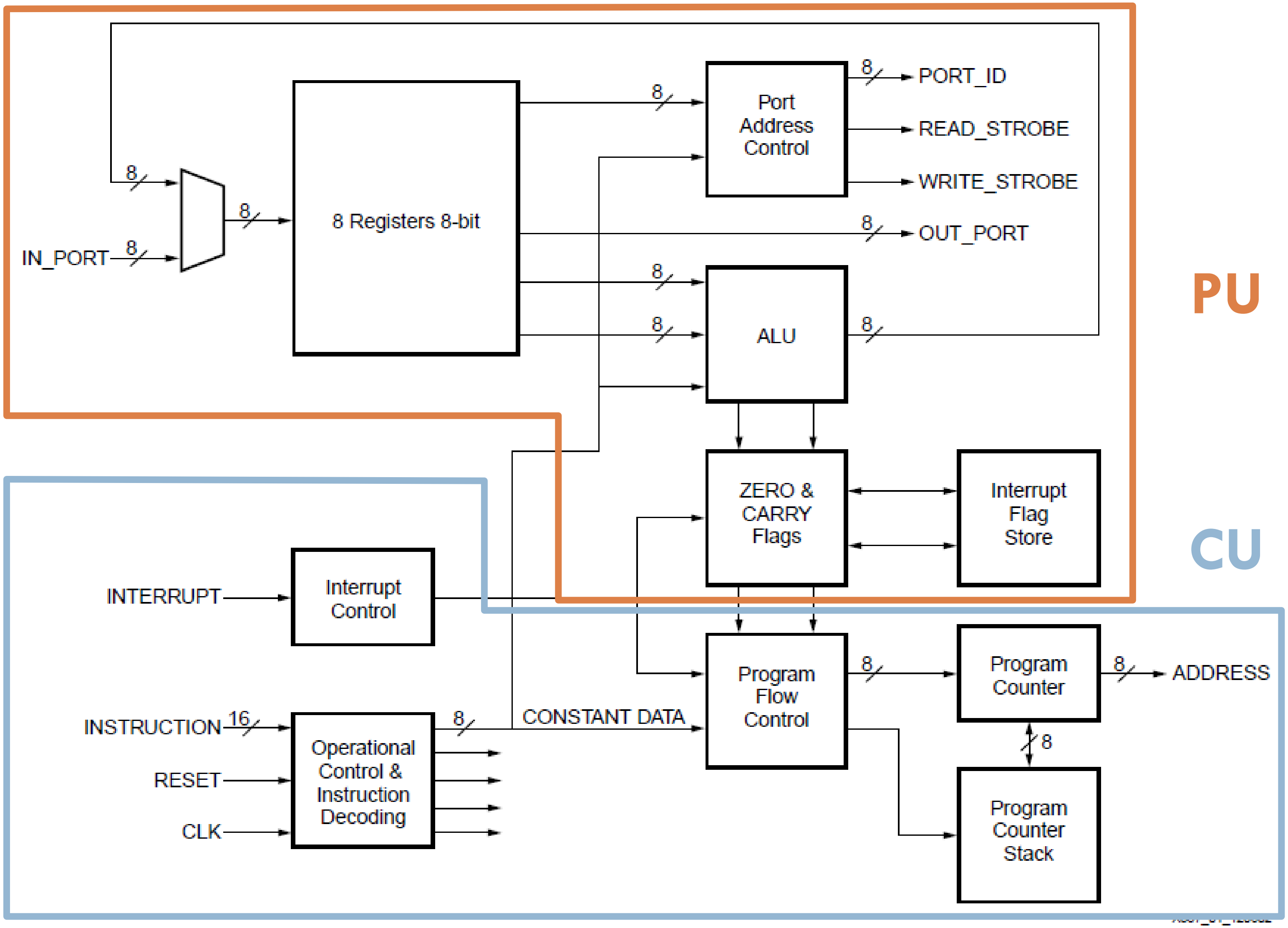
Adding more peripherals: tristated bus



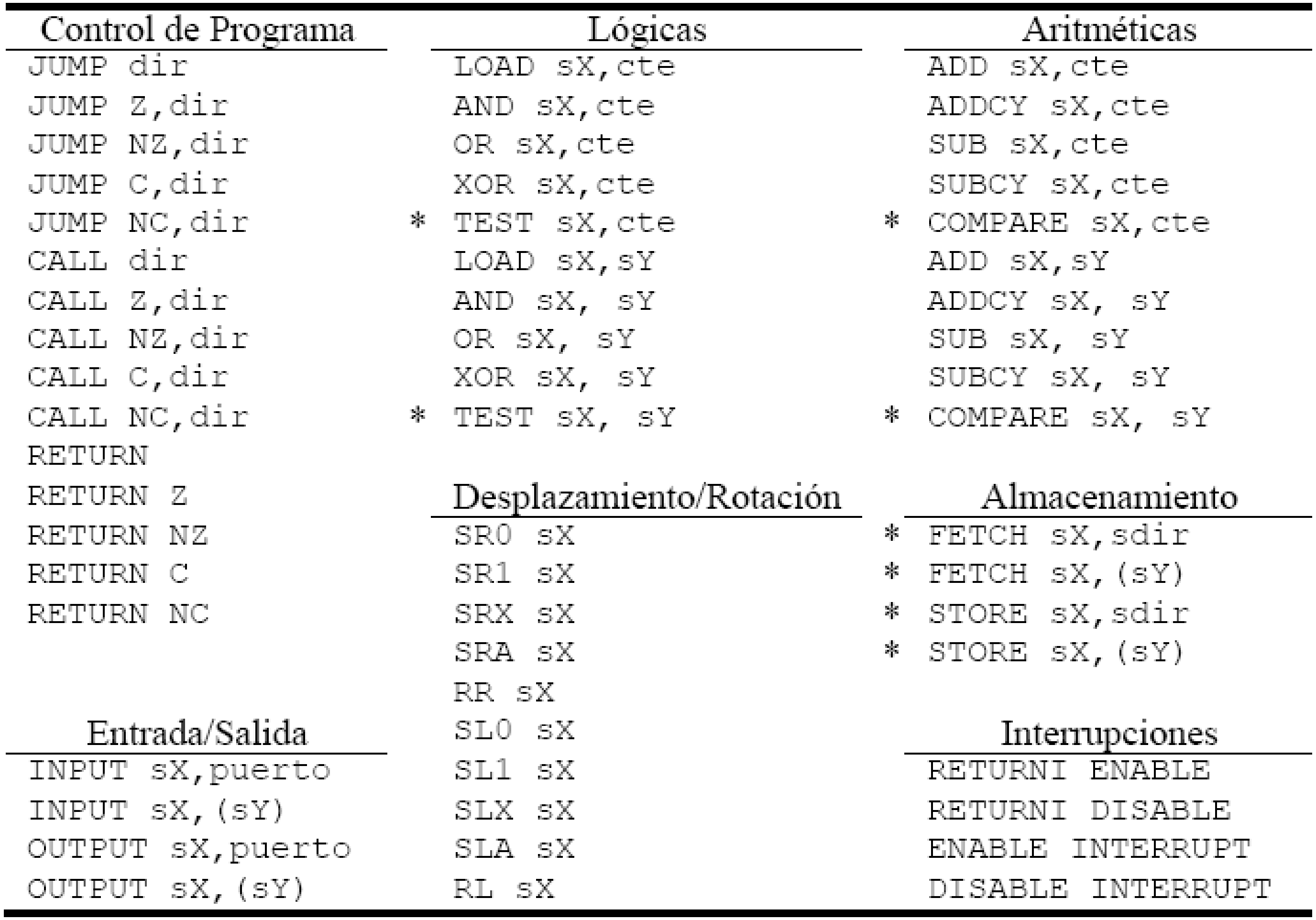
Adding more peripherals: multiplexed bus



PicoBlaze Architecture



Instruction Set

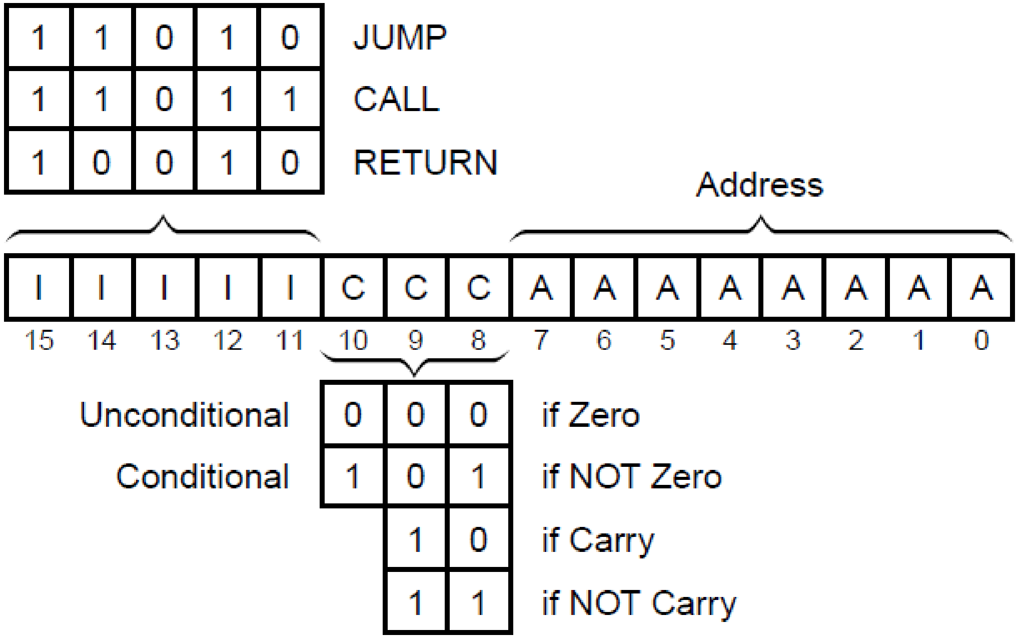


Control de Programa	Lógicas	Aritméticas
JUMP dir	LOAD sX,cte	ADD sX,cte
JUMP Z,dir	AND sX,cte	ADDCY sX,cte
JUMP NZ,dir	OR sX,cte	SUB sX,cte
JUMP C,dir	XOR sX,cte	SUBCY sX,cte
JUMP NC,dir	* TEST sX,cte	* COMPARE sX,cte
CALL dir	LOAD sX,sY	ADD sX,sY
CALL Z,dir	AND sX, sY	ADDCY sX, sY
CALL NZ,dir	OR sX, sY	SUB sX, sY
CALL C,dir	XOR sX, sY	SUBCY sX, sY
CALL NC,dir	* TEST sX, sY	* COMPARE sX, sY
RETURN		
RETURN Z	**Desplazamiento/Rotación**	**Almacenamiento**
RETURN NZ	SR0 sX	* FETCH sX,sdir
RETURN C	SR1 sX	* FETCH sX,(sY)
RETURN NC	SRX sX	* STORE sX,sdir
	SRA sX	* STORE sX,(sY)
	RR sX	
Entrada/Salida	SL0 sX	**Interrupciones**
INPUT sX,puerto	SL1 sX	RETURNI ENABLE
INPUT sX,(sY)	SLX sX	RETURNI DISABLE
OUTPUT sX,puerto	SLA sX	ENABLE INTERRUPT
OUTPUT sX,(sY)	RL sX	DISABLE INTERRUPT

Tabla 2. Juego de instrucciones del PicoBlaze. Todas las instrucciones se ejecutan en dos ciclos de reloj. Las instrucciones con (*) sólo están disponible en la versión KCPSM3 del microcontrolador.

Instruction Set Architecture

Program Control Group



1	1	0	1	0	JUMP
1	1	0	1	1	CALL
1	0	0	1	0	RETURN

I	I	I	I	I	C	C	C	A	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

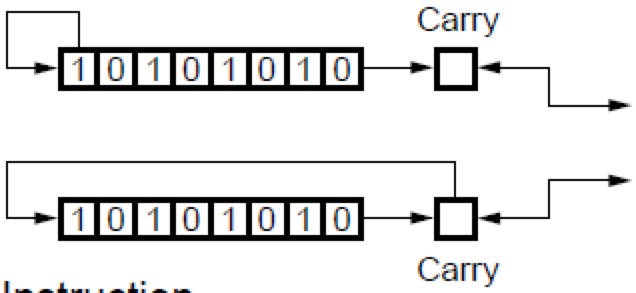
Unconditional	0	0	0	if Zero
Conditional	1	0	1	if NOT Zero
		1	0	if Carry
		1	1	if NOT Carry

JUMP aa
JUMP Z, aa
JUMP NZ, aa
JUMP C, aa
JUMP NC, aa
CALL aa
CALL Z, aa
CALL NZ, aa
CALL C, aa
CALL NC, aa
RETURN
RETURN Z
RETURN NZ
RETURN C
RETURN NC

Instruction syntax

Instruction Set Architecture

Shift and Rotate Group



SR0/SL0	1	1	0
SR1/SL1	1	1	1
SRX/SLX	0	1	0
SRA/SLA	0	0	0
RR/RL	1	0	0

Instruction

1	0	1	0	0	X	X	X	0	0	0	0	C	D	D	D
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

sX

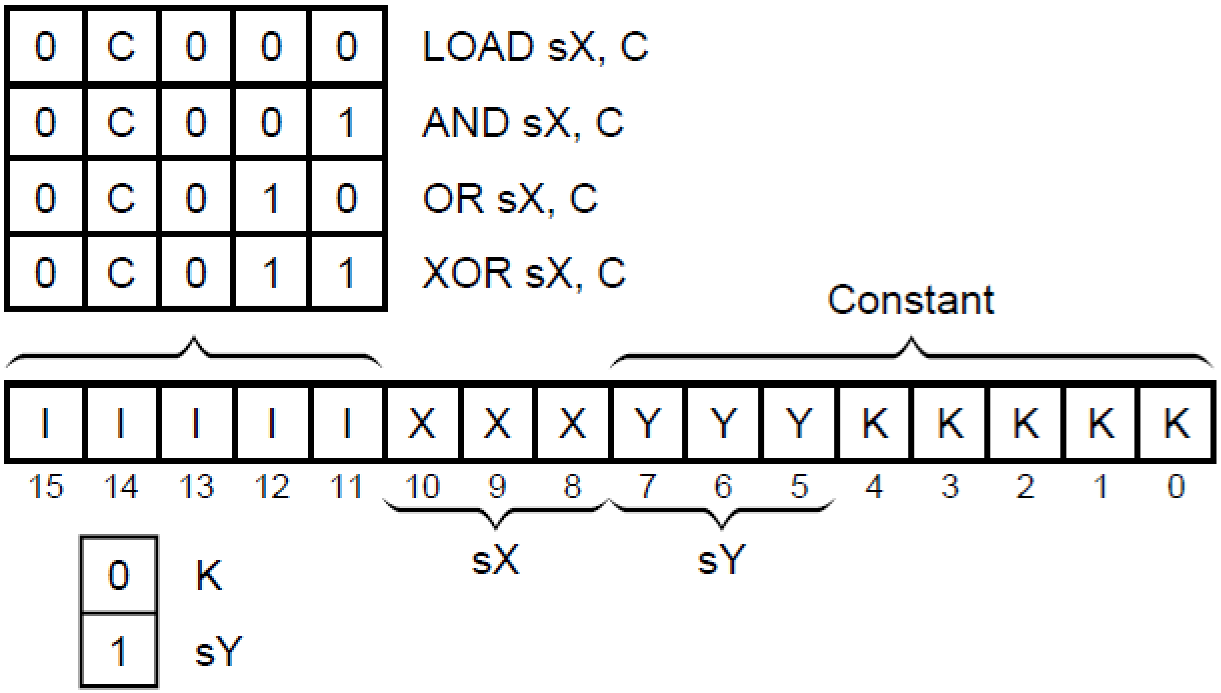
0	L - Left
1	R - Right

SR0 sX
SR1 sX
SRX sX
SRA sX
RR sX
SL0 sX
SL1 sX
SLX sX
SLA sX
RL sX

Instruction syntax

Instruction Set Architecture

Logical Group



0	C	0	0	0	LOAD sX, C
0	C	0	0	1	AND sX, C
0	C	0	1	0	OR sX, C
0	C	0	1	1	XOR sX, C

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	I	I	I	I	X	X	X	Y	Y	Y	K	K	K	K	K

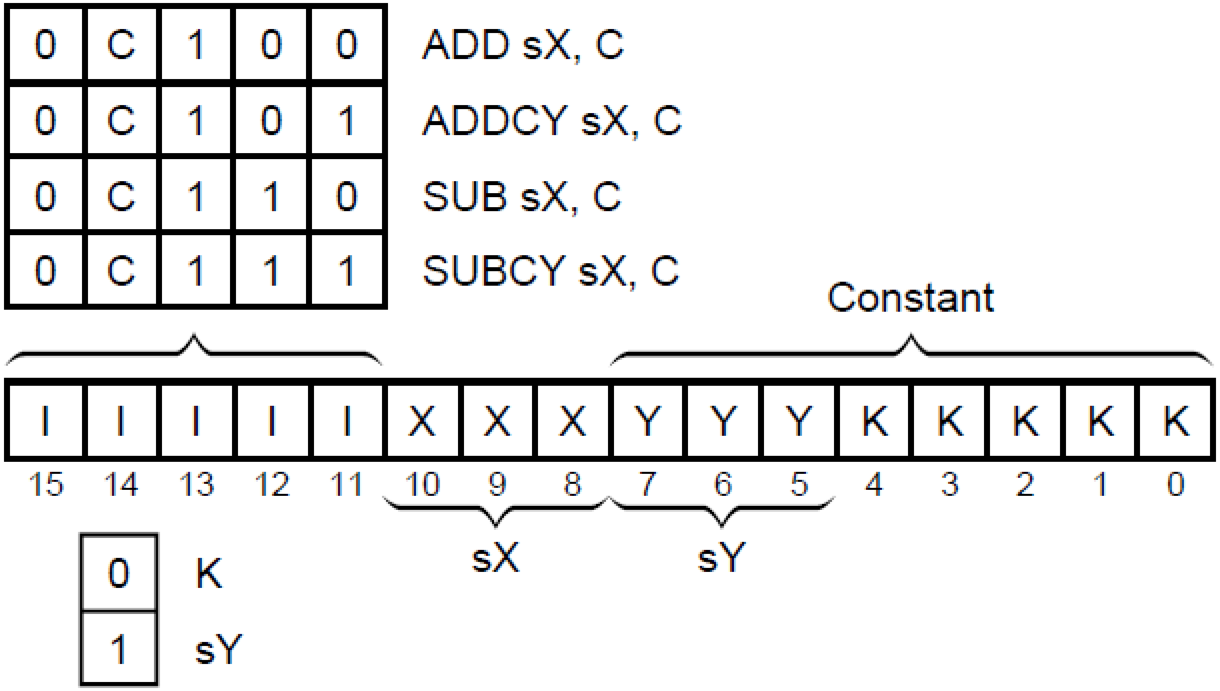
0	K
1	sY

LOAD sX, KK
AND sX, KK
OR sX, KK
XOR sX, KK
LOAD sX, sY
AND sX, sY
OR sX, sY
XOR sX, sY

Instruction syntax

Instruction Set Architecture

Arithmetic Group



0	C	1	0	0	ADD sX, C
0	C	1	0	1	ADDCY sX, C
0	C	1	1	0	SUB sX, C
0	C	1	1	1	SUBCY sX, C

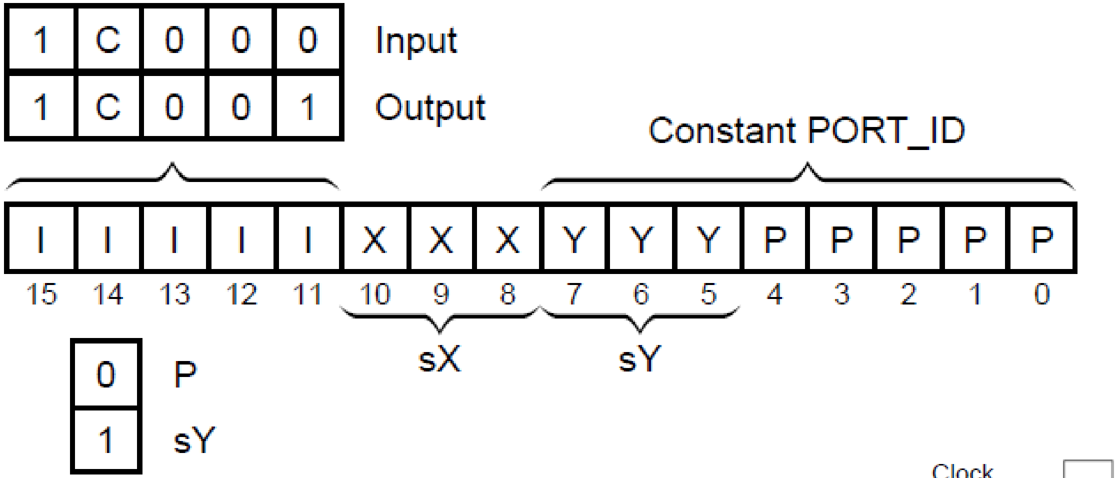
I	I	I	I	I	X	X	X	Y	Y	Y	K	K	K	K	K
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	K
1	sY

Instruction syntax
ADD sX, KK
ADDCY sX, KK
SUB sX, KK
SUBCY sX, KK
ADD sX, sY
ADDCY sX, sY
SUB sX, sY
SUBCY sX, sY

Instruction Set Architecture

Input/Output Group



1	C	0	0	0	Input
1	C	0	0	1	Output

Constant PORT_ID

I	I	I	I	I	X	X	X	Y	Y	Y	P	P	P	P	P
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

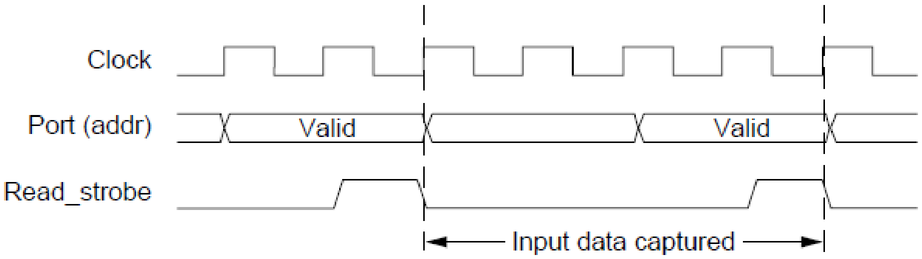
sX

sY

0	P
1	sY

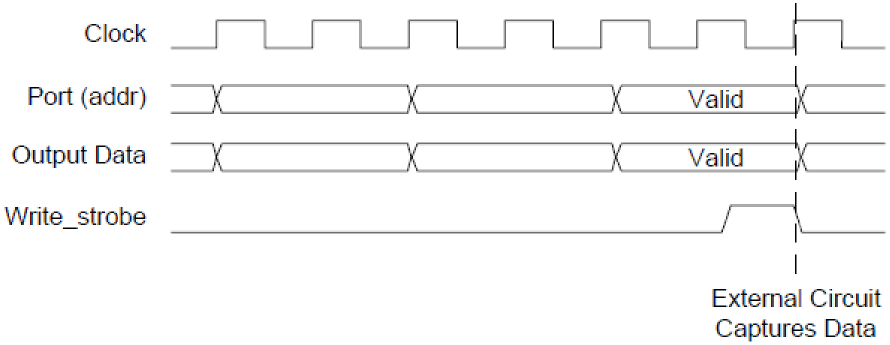
INPUT sX, PP
INPUT sX, sY
OUTPUT sX, PP
OUTPUT sX, sY

Instruction syntax



X387_07_120502

Figure 8: **Input Signal Waveform**

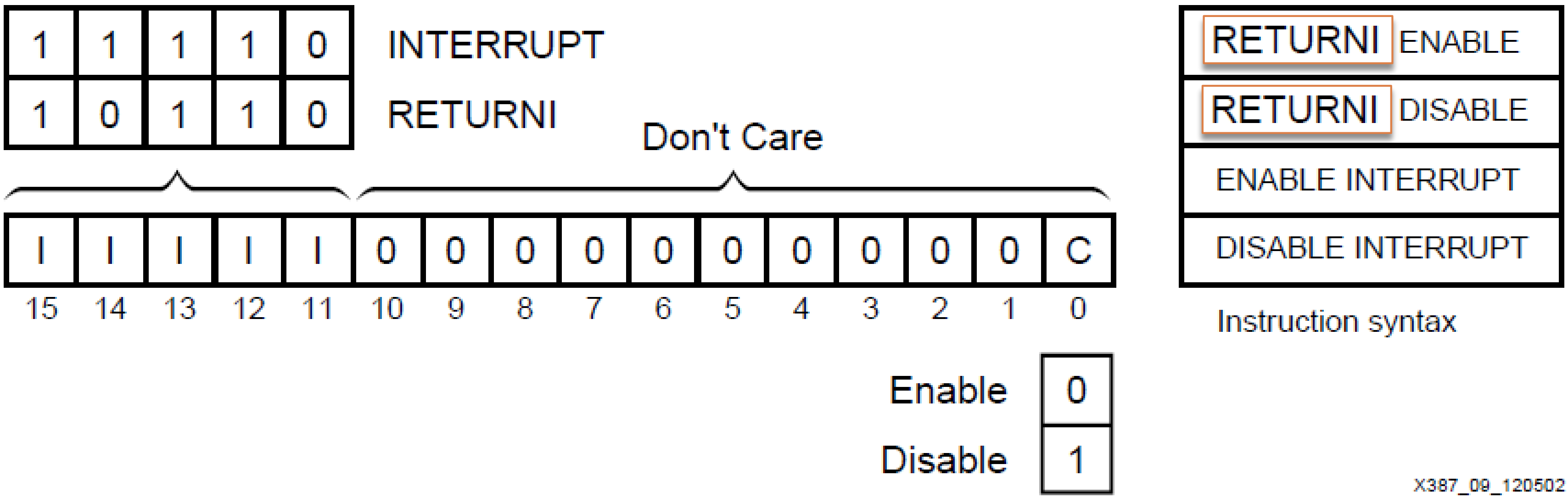


X387_08_120502

Figure 9: **Output Signal Waveform**

Instruction Set Architecture

Interrupt Group



1	1	1	1	0	INTERRUPT
1	0	1	1	0	RETURNI

Don't Care

I	I	I	I	I	0	0	0	0	0	0	0	0	0	0	C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Enable	0
Disable	1

RETURNI ENABLE

RETURNI DISABLE

ENABLE INTERRUPT

DISABLE INTERRUPT

Instruction syntax

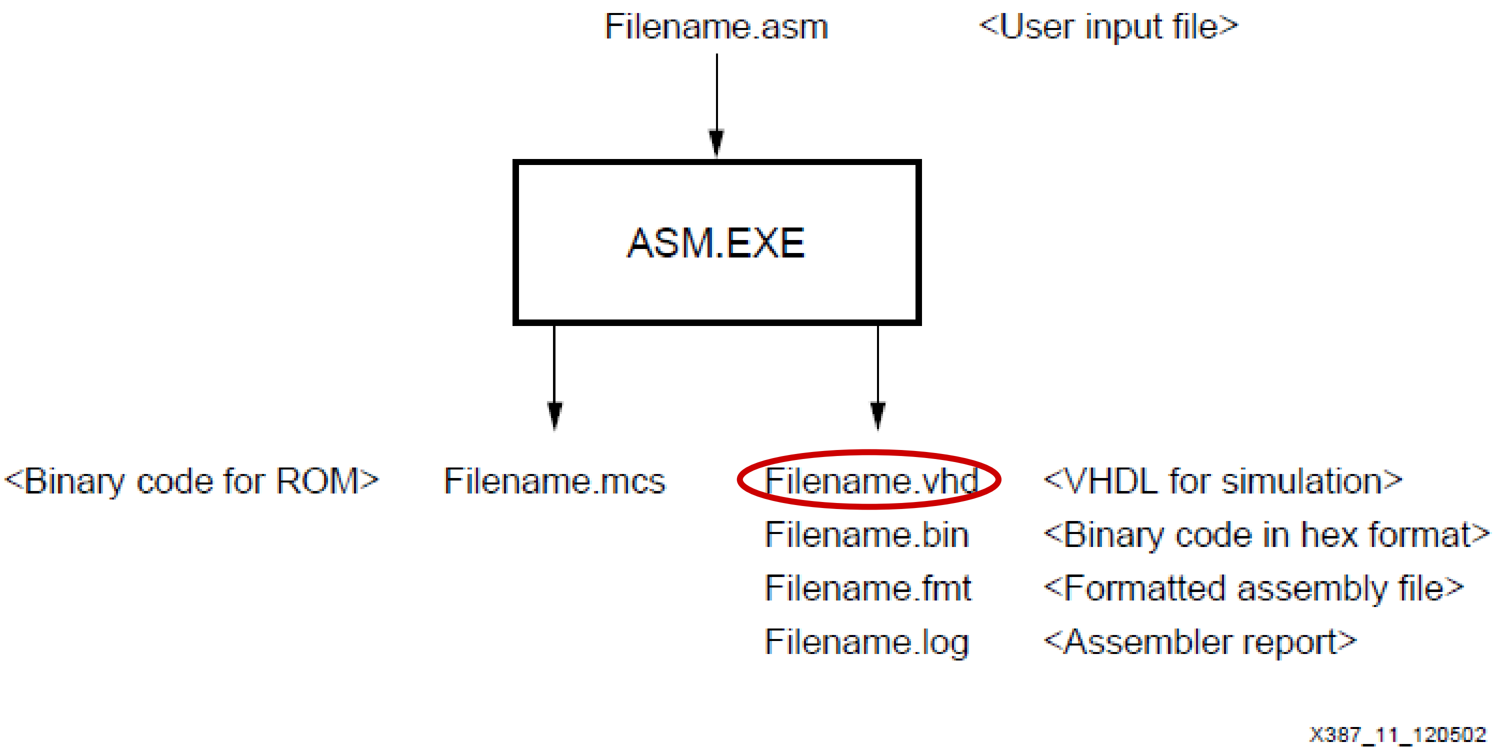
X387_09_120502

- Interrupt address vector: address FF
- There use:

  jump myintroutine
- To call interrupt routine.

Assembler: asm.exe

- MS-DOS application. Coded in C. Usage:
  - C:\>asm.exe myasmcode.asm



Program Syntax

- **No blank lines –** Use a semicolon for blank lines
- **Comments –** Any item on a line following a semicolon (;)
- **Constant** –specified in the form of a two-digit hexadecimal value (00 – FF)
- **Line Labels –** Identify program lines for JUMP or CALL instructions; should be followed by a colon (:)
- **Instructions** –Instructions and the first operand must be separated by at least one space. The assembler will accept any mixture of upper and lower case characters for the instruction.
- The assembler supports three assembler directives.
  - **CONSTANT Directive –** Assigns an 8-bit constant value to a label
  - **NAMEREG Directive –** Assigns a new name to any of the eight registers
  - **ADDRESS Directive –** Forces the instructions that follow it to commence at a new address value.

Program Syntax: “hello world” example

```
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;declaracion de constantes y variables
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         CONSTANT         rs232, 00; puerto comunicacion serie es el 00
                                   ; rx es el bit 0 del puerto 00(entrada)
                                   ; tx es el bit 7 del puerto 00(salida),
         ;porque hyperterminal envia primero el LSB, por eso desplazamoa a la
         ;izquierda al recibir, y al enviar, con lo que enviamos de nuevo
         ;el LSB primero como corresponde para que lo entienda hyperterminal
         NAMEREG          s1, txreg        ;buffer de transmision
         NAMEREG          s2, rxreg        ;buffer de recepcion
         NAMEREG          s3, contbit      ;contador de los 8 bits de datos
         NAMEREG          s4, cont1        ;contador de retardo1
         NAMEREG          s5, cont2        ;contador de retardo2
         ;
         ADDRESS          00       ; programa se cargara comenzando en dir 00
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;Inicio del programa
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:
         ...
         ...
```

Program Syntax: “hello world” example

```
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;Inicio del programa
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:   ;esperamos a recibir un caracter
         CALL     recibe
         ;copiamos el caracter recibido al buffer de transmision
         LOAD     txreg, rxreg
         ADD      txreg, 01
         ;hacemos el eco del caracter recibido
         CALL     transmite
         JUMP     start
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;Rutina de recepcion de caracteres
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

         ...
         ...
```

Program Syntax: “hello world” example

```
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;Rutina de recepcion de caracteres
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
recibe:          ;esperamos a que se reciba un bit de inicio
                 INPUT    rxreg, rs232
                 AND      rxreg, 80
                 JUMP     NZ, recibe
                 CALL     wait_05bit
                 ;almacenamos los 8 bits de datos
                 LOAD     contbit,09
next_rx_bit:     CALL             wait_1bit
                 SR0      rxreg
                 INPUT    s0, rs232
                 AND      s0, 80
                 OR       rxreg, s0
                 SUB      contbit, 01
                 JUMP     NZ, next_rx_bit
                 RETURN
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;Rutina de transmision de caracteres
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ...
                 ...
```

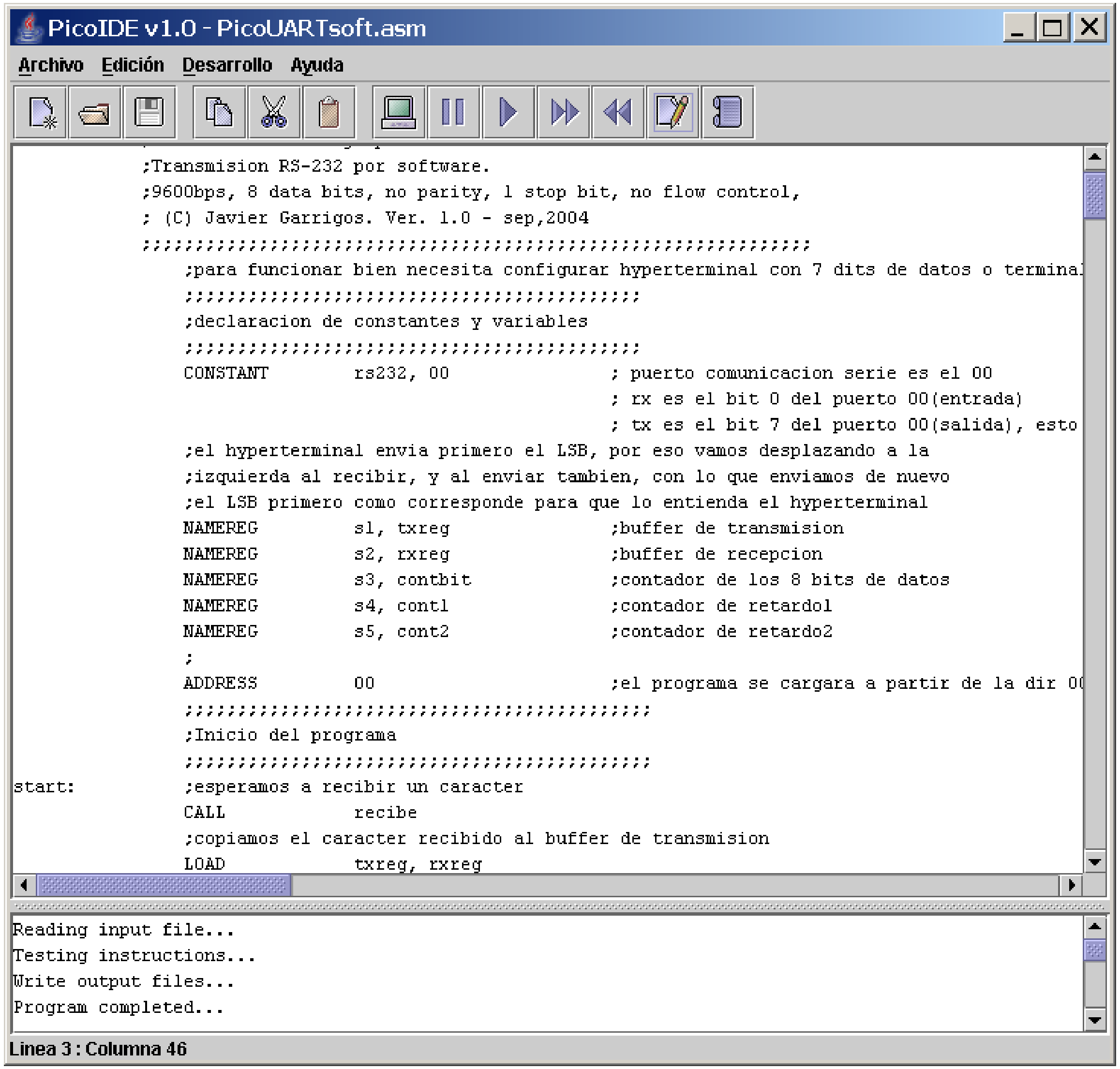
Program Syntax: “hello world” example

```
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;Rutina de transmision de caracteres
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
transmite:       ;enviamos un bit de inicio
                 LOAD             s0, 00
                 OUTPUT           s0, rs232
                 CALL             wait_1bit
                 ;enviamos los 8 bits de datos
                 LOAD             contbit, 08
next_tx_bit:     OUTPUT           txreg, rs232
                 CALL             wait_1bit
                 SR0              txreg
                 SUB              contbit, 01
                 JUMP             NZ, next_tx_bit
                 ;enviamos un bit de parada
                 LOAD             s0, FF
                 OUTPUT           s0, rs232
                 CALL             wait_1bit
                 RETURN
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ;Rutina espera 1 bit (a 115200bps)
                 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                 ...
                 ...
```

Program Syntax: “hello world” example

```
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;Rutina espera 1 bit (a 115200bps)
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;clk=50MHz, 115200bps, cont1=03, cont2=22
         ;esta rutina ejecuta 1+(1+3*(1+34*2+2))+1 = 216 instruciones,
         ;aproximandose al numero teorico de
         ;8,68 us/bit)/(0,04 us/instruc) = 217 instr/bit necesarias.
wait_1bit:       LOAD             cont1, 03
espera2:         LOAD             cont2, 22
espera1:         SUB              cont2, 01
                 JUMP             NZ, espera1
                 SUB              cont1, 01
                 JUMP             NZ, espera2
                 RETURN
                 ...
                 ...
```

PicoIDE: assembler and debugger



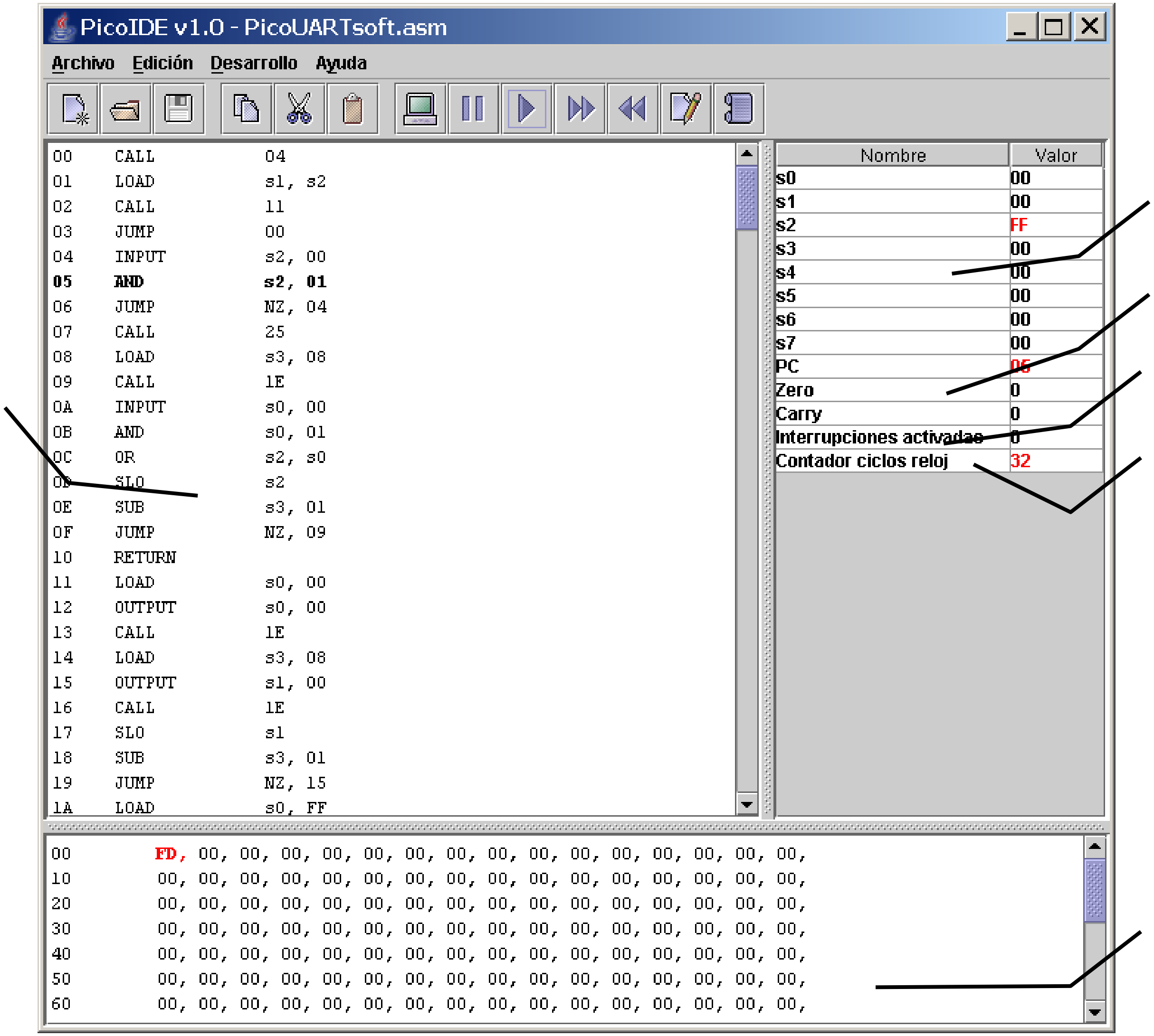
```
            ;Transmision RS-232 por software.
            ;9600bps, 8 data bits, no parity, 1 stop bit, no flow control,
            ; (C) Javier Garrigos. Ver. 1.0 - sep,2004
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;para funcionar bien necesita configurar hyperterminal con 7 dits de datos o terminal
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;declaracion de constantes y variables
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                CONSTANT        rs232, 00                ; puerto comunicacion serie es el 00
                                                         ; rx es el bit 0 del puerto 00(entrada)
                                                         ; tx es el bit 7 del puerto 00(salida), esto
                ;el hyperterminal envia primero el LSB, por eso vamos desplazando a la
                ;izquierda al recibir, y al enviar tambien, con lo que enviamos de nuevo
                ;el LSB primero como corresponde para que lo entienda el hyperterminal
                NAMEREG         s1, txreg                ;buffer de transmision
                NAMEREG         s2, rxreg                ;buffer de recepcion
                NAMEREG         s3, contbit              ;contador de los 8 bits de datos
                NAMEREG         s4, cont1                ;contador de retardo1
                NAMEREG         s5, cont2                ;contador de retardo2
                ;
                ADDRESS         00                       ;el programa se cargara a partir de la dir 00
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;Inicio del programa
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:          ;esperamos a recibir un caracter
                CALL            recibe
                ;copiamos el caracter recibido al buffer de transmision
                LOAD            txreg, rxreg
```

```
Reading input file...
Testing instructions...
Write output files...
Program completed...
```

Linea 3 : Columna 46

- ❑ Java app
- ❑ Graphical IDE
- ❑ Developed at the UPCT
- ❑ by students like you!!
- ❑ Two modes:
  - ❑ Text editing and assembling
  - ❑ Debugging
- ❑ Optional (typically just for debugging)

PicoIDE: assembler and debugger



```
00    CALL      04
01    LOAD      s1, s2
02    CALL      11
03    JUMP      00
04    INPUT     s2, 00
05    AND       s2, 01
06    JUMP      NZ, 04
07    CALL      25
08    LOAD      s3, 08
09    CALL      1E
0A    INPUT     s0, 00
0B    AND       s0, 01
0C    OR        s2, s0
0D    SL0       s2
0E    SUB       s3, 01
0F    JUMP      NZ, 09
10    RETURN
11    LOAD      s0, 00
12    OUTPUT    s0, 00
13    CALL      1E
14    LOAD      s3, 08
15    OUTPUT    s1, 00
16    CALL      1E
17    SL0       s1
18    SUB       s3, 01
19    JUMP      NZ, 15
1A    LOAD      s0, FF
```

Nombre	Valor
s0	00
s1	00
s2	FF
s3	00
s4	00
s5	00
s6	00
s7	00
PC	05
Zero	0
Carry	0
Interrupciones activadas	0
Contador ciclos reloj	32

```
00      FD, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
10      00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
20      00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
30      00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
40      00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
50      00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
60      00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,
```

Disassembled code

Internal Registers

Status Reg Flags

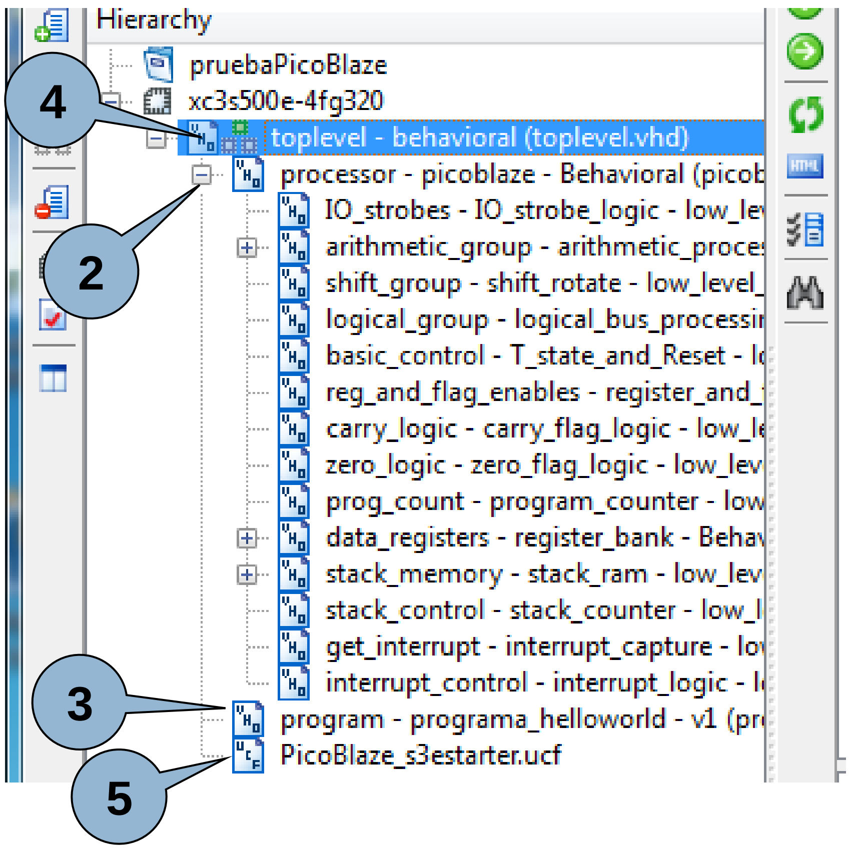
Interrupt Flag

Program Counter

I/O Interfaces

Gathering it all!!!

1. Create a new ISE project
2. Add source vhdl code for PicoBlaze
3. Add vhdl code obtained from assembler for the IRAM (containing PB app program)
4. Add a toplevel entity and instantiate PB, IRAM, peripherals, etc. as desired
5. Add User Constraints File (.ucf) with clk and pin specification.
6. Optionally, add vhdl code for a testbench
7. Cross your fingers…



Hierarchy

pruebaPicoBlaze

xc3s500e-4fg320

toplevel - behavioral (toplevel.vhd)

processor - picoblaze - Behavioral (picob

IO_strobes - IO_strobe_logic - low_lev

arithmetic_group - arithmetic_proce

shift_group - shift_rotate - low_level_

logical_group - logical_bus_processin

basic_control - T_state_and_Reset - lo

reg_and_flag_enables - register_and_

carry_logic - carry_flag_logic - low_le

zero_logic - zero_flag_logic - low_lev

prog_count - program_counter - low

data_registers - register_bank - Behav

stack_memory - stack_ram - low_lev

stack_control - stack_counter - low_l

get_interrupt - interrupt_capture - lov

interrupt_control - interrupt_logic - l

program - programa_helloworld - v1 (pr

PicoBlaze_s3estarter.ucf