Von der Hardware zur Software in FPGAs mit Embedded Prozessoren

Alexander Hahn
Senior Field Application Engineer
Lattice Semiconductor

AGENDA

Overview

Mico32 Embedded Processor

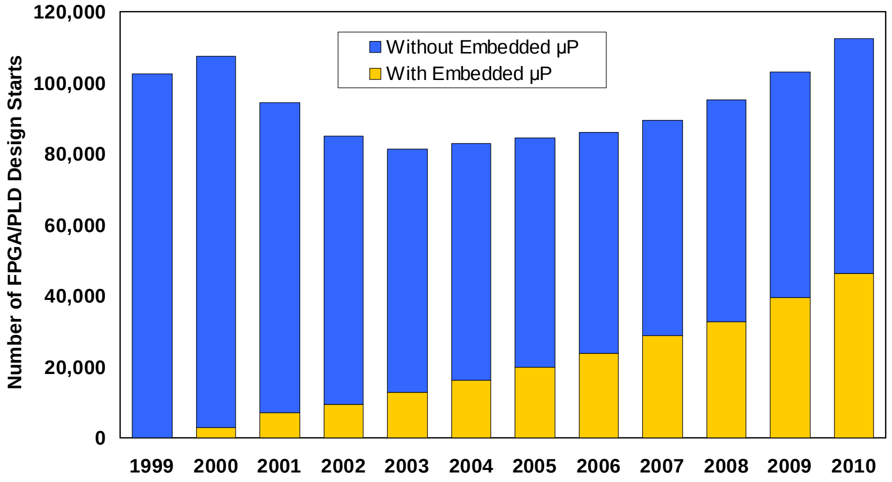
Development Tool Chain

HW/SW Example

Conclusion

Trend Towards Embedded Processors

- Embedded Processor well suited to implement control flow based applications
- Programmable solution allows more flexibility for the design
- On-chip processor gains close interaction with the dedicated HW logic
- Application specific HW modules can be integrated into processor & SW environment
- Integrated development tools allow fast design cycle time
- Open source & soft IP avoid obsolescence



AGENDA

Overview

Mico32 Embedded Processor

Development Tool Chain

HW/SW Example

Conclusion

The Lattice Mico32 Embedded RISC Processor Core

- **Innovative Open IP Soft Core License**
  - **No Fees or Royalties**
  - **Provides visibility, flexibility and portability**

- **Performance Enhanced Features**
  - **Load/Store RISC Architecture**
  - **Harvard Architecture**
    - **32-bit Data Path**
    - **32-bit Instruction Path**
  - **32 General Purpose Registers**
  - **Handles Up to 32 External Interrupts**
  - **Optional Instruction and/or Data Cache**
  - **OpenCores WISHBONE Compatible Busses**
  - **DMA controller**

- **Rich Peripheral Selection**

- **Easy to Use Development Tools**
  - **Platform definition**
  - **C/C++ software development**
  - **Debug**

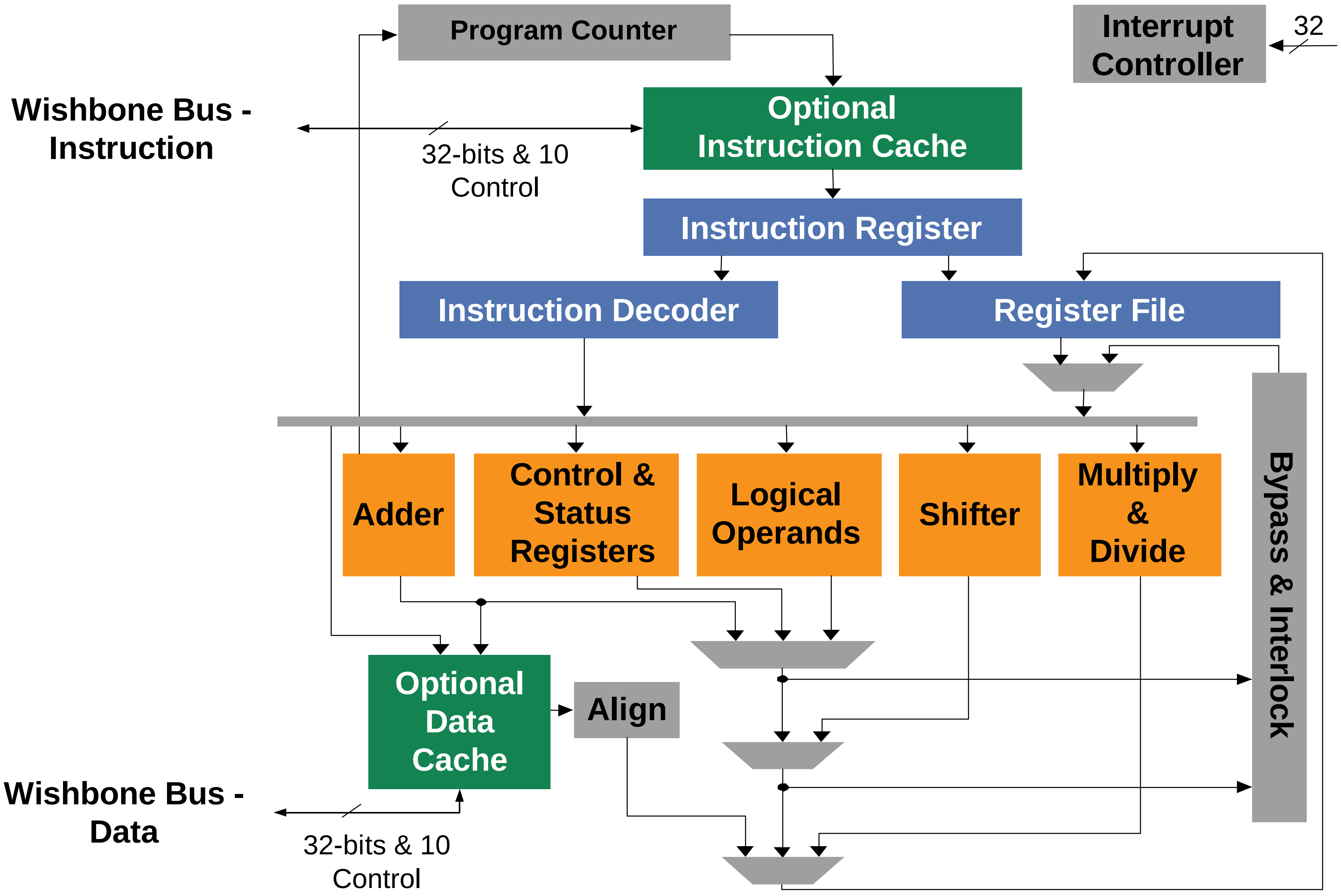
- **Interconnect**
  - 32 bit Wishbone instruction & data bus
  - Shared bus arbiter

- **Memory**
  - On Chip Memory
  - Parallel Flash
  - Async SRAM
  - DDR1/DDR2
  - SDRAM
  - SPI Flash

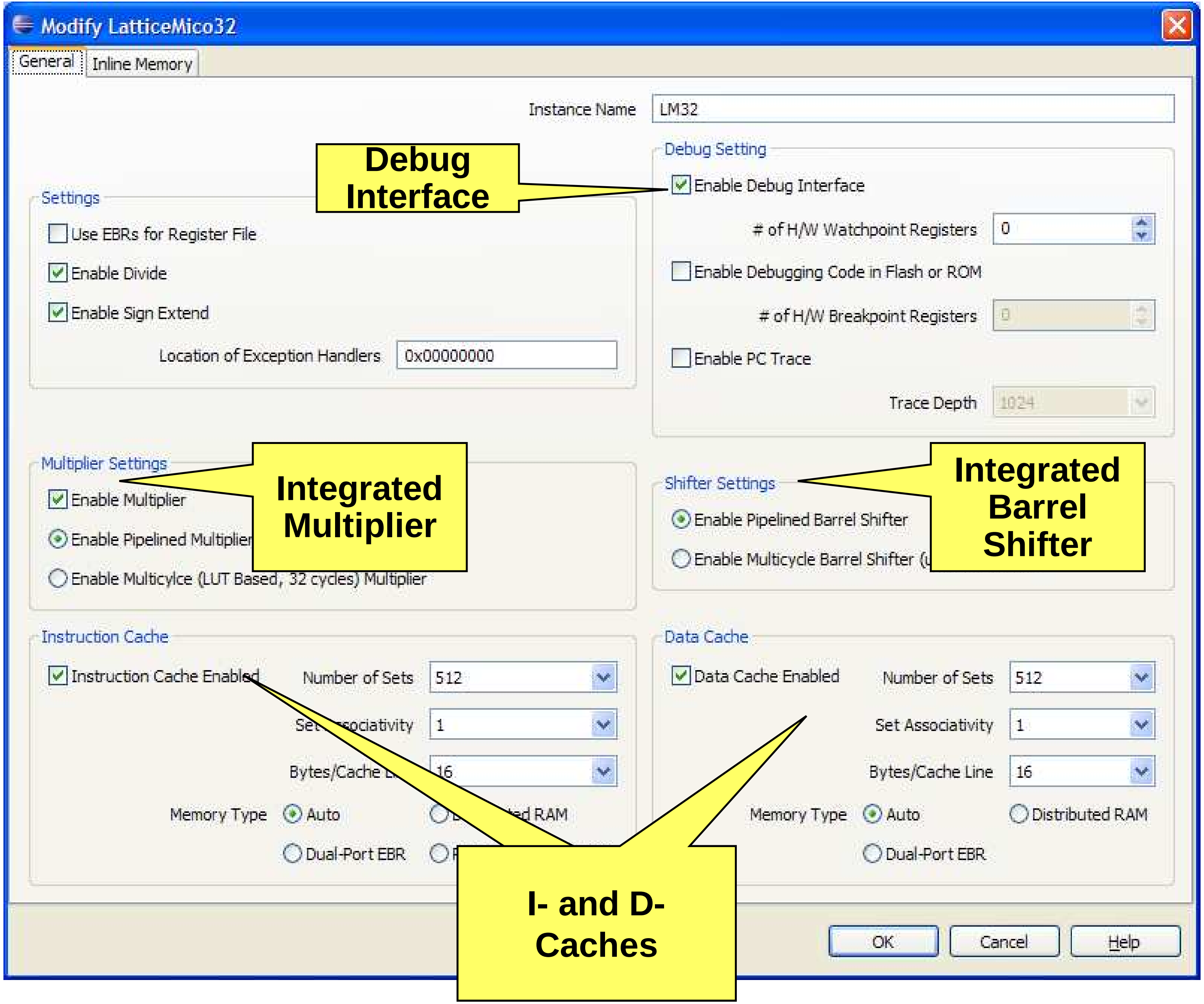
- **Peripherals Components**
  - DMA Unit
  - GPIO, Timer, UART, SPI
  - I2C Master (Opencores)
  - Tri-Speed Ethernet MAC
  - PCI Target Controller
  - Custom User Components

- **Custom Extensions**
  - “Write your Own”

Mico32 Arcitecture



Mico32 Configuration Options



Modify LatticeMico32

General | Inline Memory

Instance Name: LM32

Settings

- ☐ Use EBRs for Register File
- ☑ Enable Divide
- ☑ Enable Sign Extend
- Location of Exception Handlers: 0x00000000

Debug Setting

- ☑ Enable Debug Interface
- # of H/W Watchpoint Registers: 0
- ☐ Enable Debugging Code in Flash or ROM
- # of H/W Breakpoint Registers: 0
- ☐ Enable PC Trace
- Trace Depth: 1024

Multiplier Settings

- ☑ Enable Multiplier
- ● Enable Pipelined Multiplier
- ○ Enable Multicylce (LUT Based, 32 cycles) Multiplier

Shifter Settings

- ● Enable Pipelined Barrel Shifter
- ○ Enable Multicycle Barrel Shifter (u

Instruction Cache

- ☑ Instruction Cache Enabled
- Number of Sets: 512
- Set Associativity: 1
- Bytes/Cache L: 16
- Memory Type: ● Auto ○ D ed RAM ○ Dual-Port EBR ○ P

Data Cache

- ☑ Data Cache Enabled
- Number of Sets: 512
- Set Associativity: 1
- Bytes/Cache Line: 16
- Memory Type: ● Auto ○ Distributed RAM ○ Dual-Port EBR

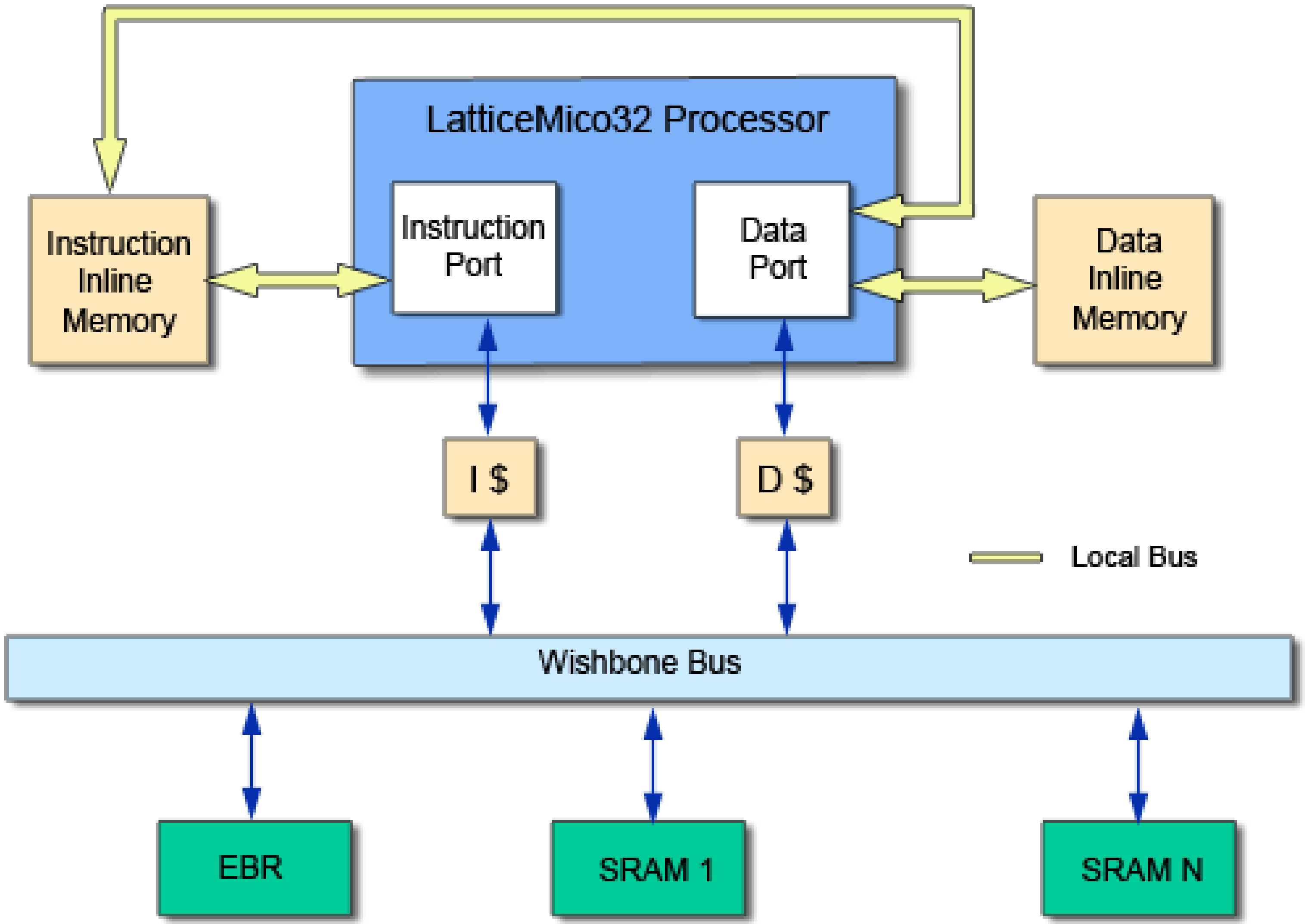
Debug Interface

Integrated Multiplier

Integrated Barrel Shifter

I- and D-Caches

Mico32 Memory System

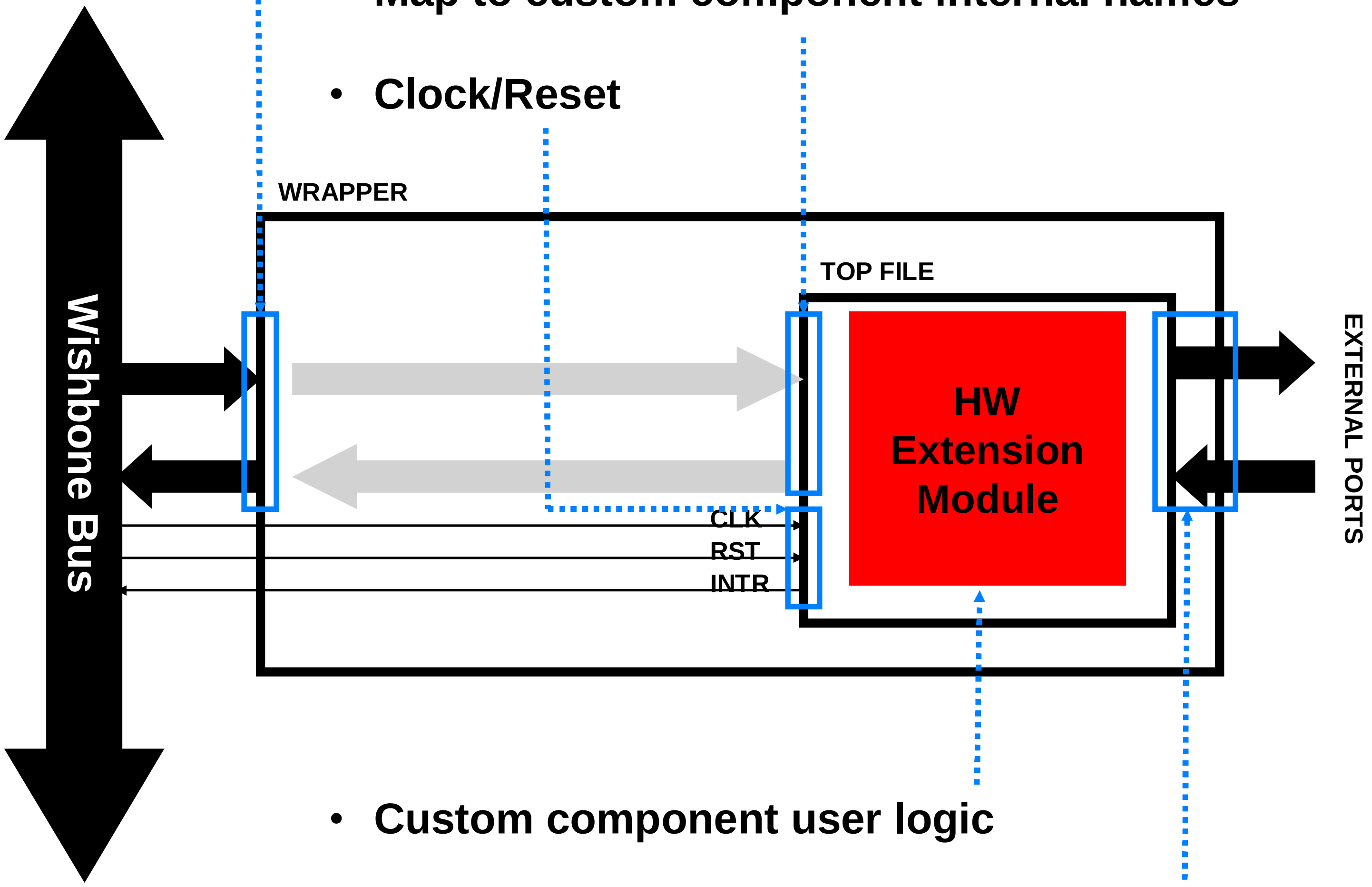


Combine Processor Core with Custom Components

- Mico32 architecture can be enhanced by custom components
  - Custom component map to wishbone bus interface
  - Easy to integrate via GUI wizard
  - Write to/read from external components
  - Components can be used as co-processing units to boost performance
    - Component mapped into memory or IO space of Mico32
    - Synchronization via interrupt or polling
    - Write to registers may perform side effects
    - Examples:
      - CRC calculation
      - Data extraction from data stream
      - Enhance functionality by DSP functions, memories, logic.
  - Direct interconnection with FPGA fabric logic
  - SW access handled via templates

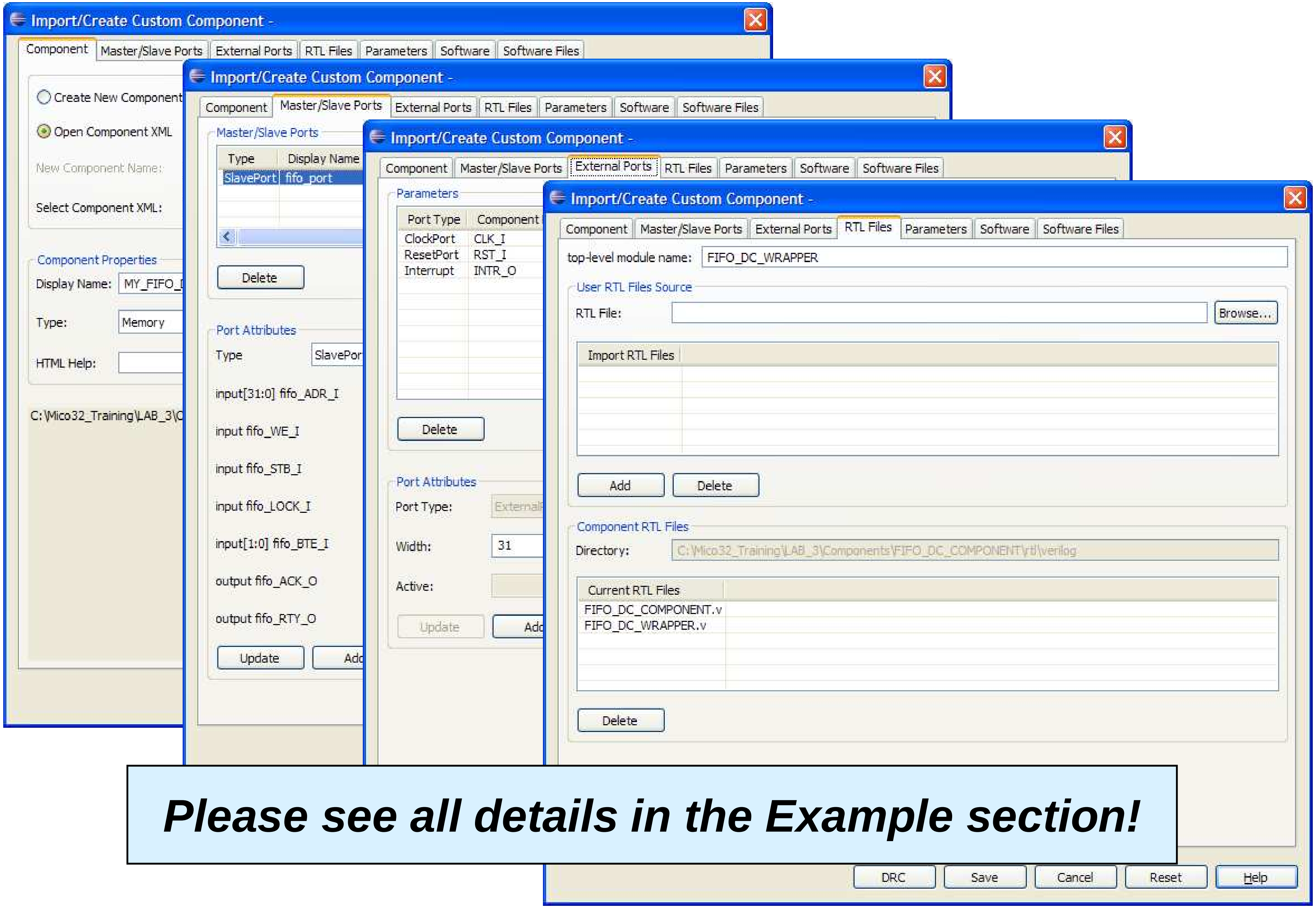
Integration of Custom Components

- Wishbone bus signals
- Map to custom component internal names
- Clock/Reset



- Custom component user logic
- Custom component user signals exported to external interface

GUI for Custom Component Creation



○ Create New Component

● Open Component XML

New Component Name:

Select Component XML:

Component Properties

Display Name: MY_FIFO_[

Type: Memory

HTML Help:

C:\Mico32_Training\LAB_3\C

Master/Slave Ports

Type	Display Name
SlavePort	fifo_port

Port Attributes

Type SlavePor

input[31:0] fifo_ADR_I

input fifo_WE_I

input fifo_STB_I

input fifo_LOCK_I

input[1:0] fifo_BTE_I

output fifo_ACK_O

output fifo_RTY_O

Parameters

Port Type	Component
ClockPort	CLK_I
ResetPort	RST_I
Interrupt	INTR_O

Port Attributes

Port Type: External

Width: 31

Active:

top-level module name: FIFO_DC_WRAPPER

User RTL Files Source

RTL File:

Component RTL Files

Directory: C:\Mico32_Training\LAB_3\Components\FIFO_DC_COMPONENT\rtl\verilog

Current RTL Files
FIFO_DC_COMPONENT.v
FIFO_DC_WRAPPER.v

Please see all details in the Example section!

AGENDA

Overview

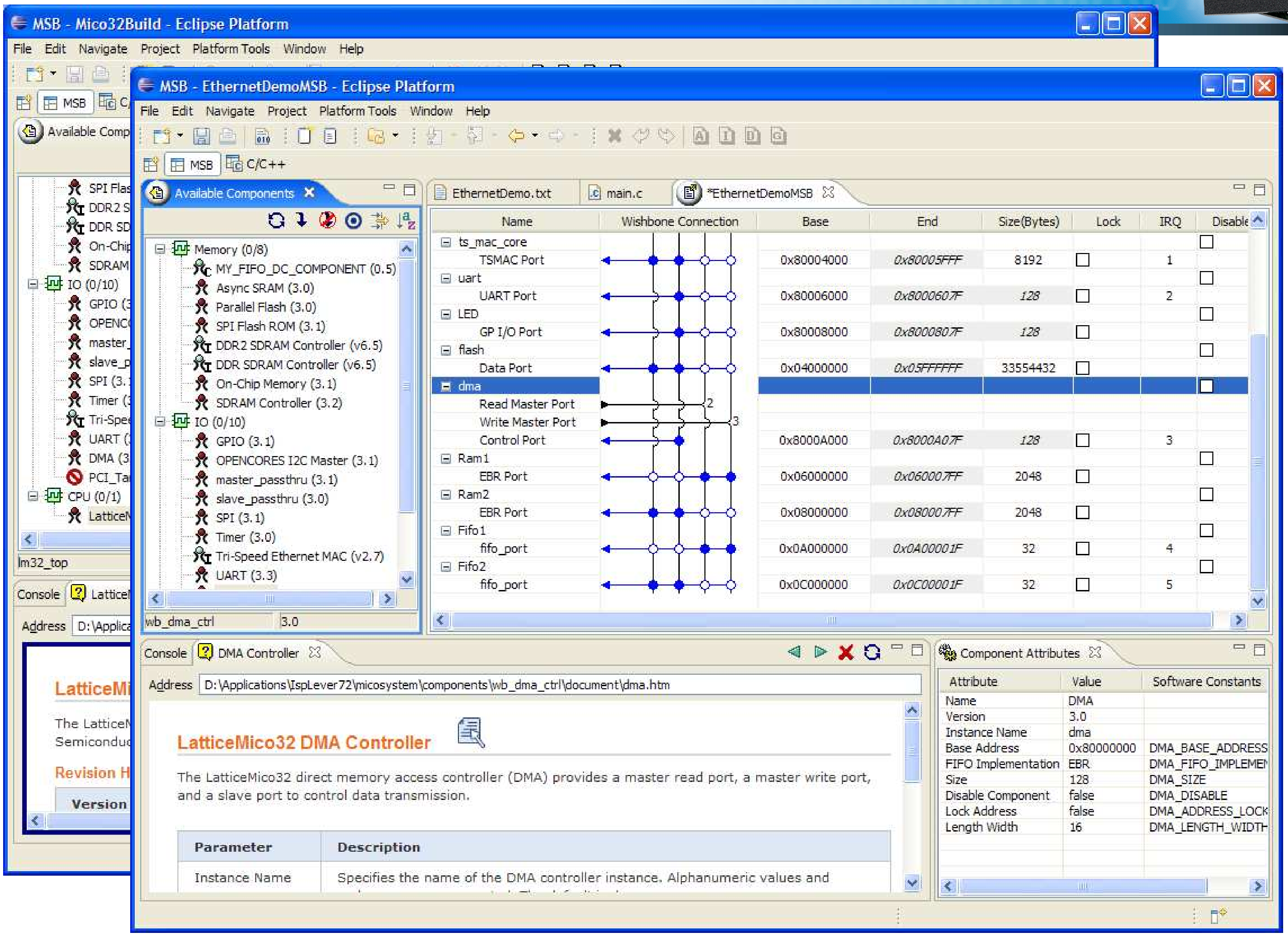
Mico32 Embedded Processor

Development Tool Chain

HW/SW Example

Conclusion

Mico32 System Builder



Name	Wishbone Connection	Base	End	Size(Bytes)	Lock	IRQ	Disable
ts_mac_core							□
TSMAC Port		0x80004000	0x80005FFF	8192	□	1	
uart							□
UART Port		0x80006000	0x8000607F	128	□	2	
LED							□
GP I/O Port		0x80008000	0x8000807F	128	□		
flash							□
Data Port		0x04000000	0x05FFFFFF	33554432	□		
dma							□
Read Master Port	2						
Write Master Port	3						
Control Port		0x8000A000	0x8000A07F	128	□	3	
Ram1							□
EBR Port		0x06000000	0x060007FF	2048	□		
Ram2							□
EBR Port		0x08000000	0x080007FF	2048	□		
Fifo1							□
fifo_port		0x0A000000	0x0A00001F	32	□	4	
Fifo2							□
fifo_port		0x0C000000	0x0C00001F	32	□	5	

Address D:\Applications\IspLever72\micosystem\components\wb_dma_ctrl\document\dma.htm

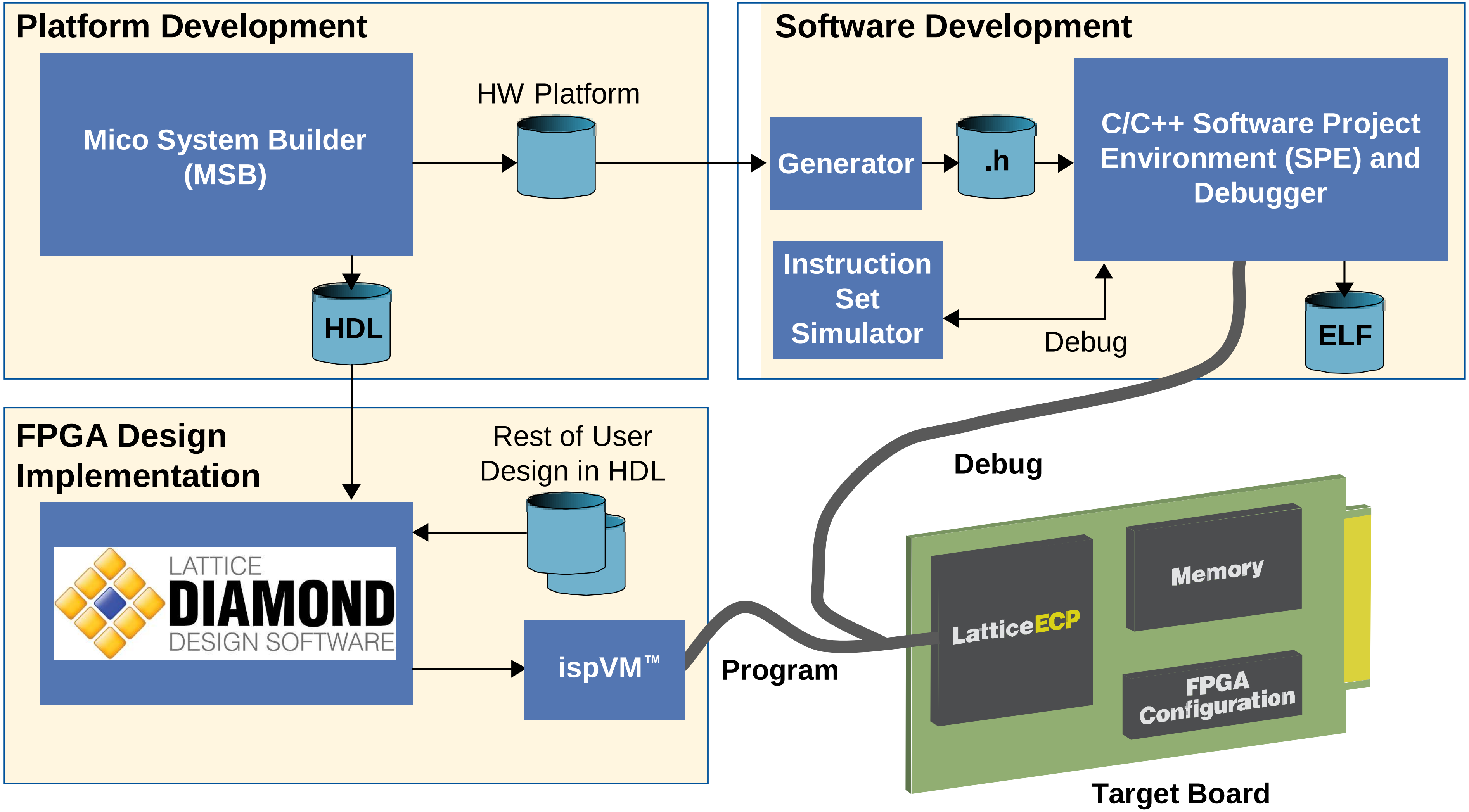
LatticeMico32 DMA Controller

The LatticeMico32 direct memory access controller (DMA) provides a master read port, a master write port, and a slave port to control data transmission.

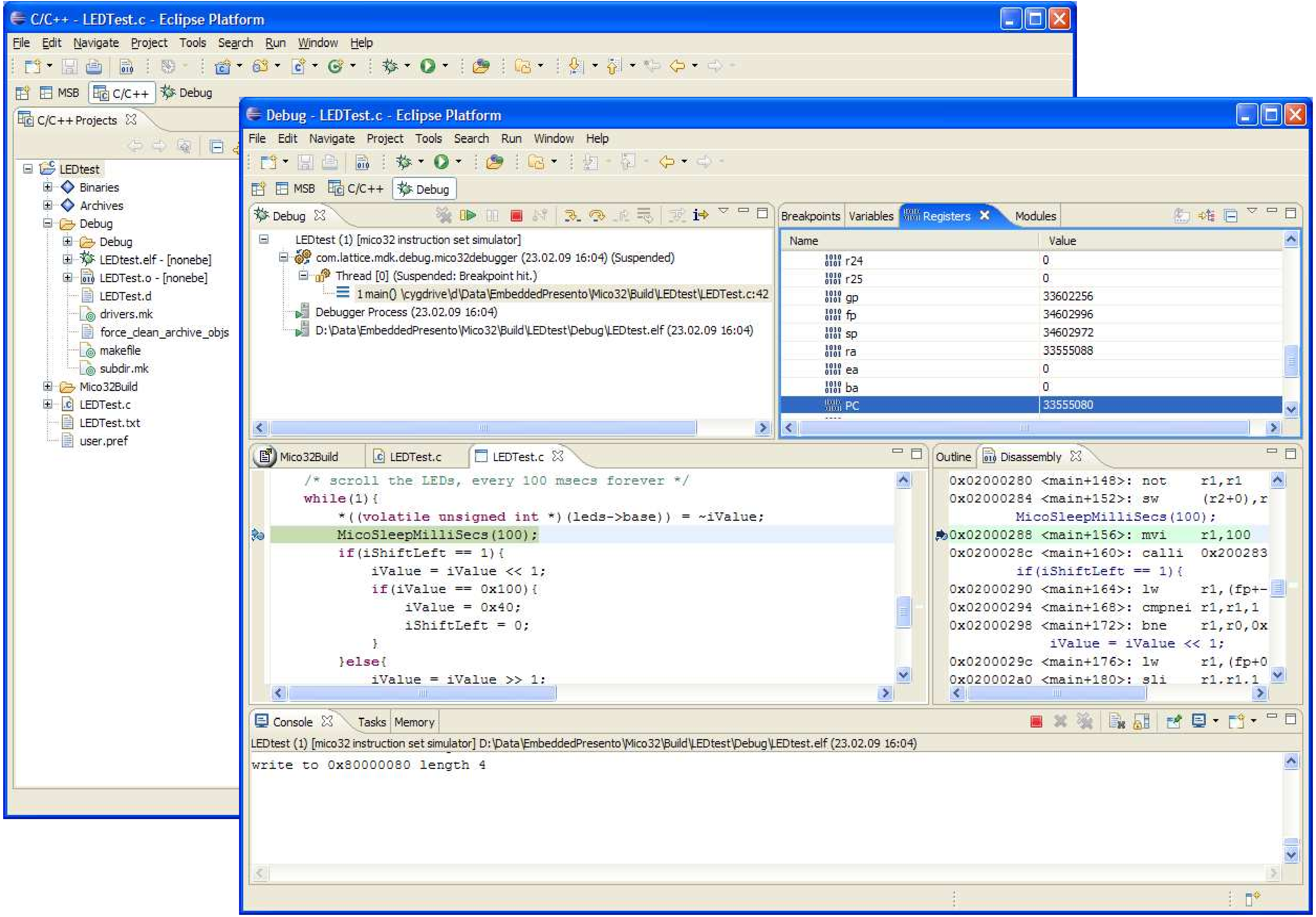
Parameter	Description
Instance Name	Specifies the name of the DMA controller instance. Alphanumeric values and

Attribute	Value	Software Constants
Name	DMA	
Version	3.0	
Instance Name	dma	
Base Address	0x80000000	DMA_BASE_ADDRESS
FIFO Implementation	EBR	DMA_FIFO_IMPLEMEN
Size	128	DMA_SIZE
Disable Component	false	DMA_DISABLE
Lock Address	false	DMA_ADDRESS_LOCK
Length Width	16	DMA_LENGTH_WIDTH

Build HW - Write SW - Evaluation Board - Debug


Platform Development
Mico System Builder (MSB)
HW Platform
HDL
Software Development
Generator
.h
C/C++ Software Project Environment (SPE) and Debugger
Instruction Set Simulator
Debug
ELF
FPGA Design Implementation
Rest of User Design in HDL
LATTICE DIAMOND DESIGN SOFTWARE
ispVM™
Program
Debug
LatticeECP
Memory
FPGA Configuration
Target Board

SW Development & Debug Environment


C/C++ - LEDTest.c - Eclipse Platform
File Edit Navigate Project Tools Search Run Window Help
MSB C/C++ Debug
C/C++ Projects
Debug - LEDTest.c - Eclipse Platform
File Edit Navigate Project Tools Search Run Window Help
MSB C/C++ Debug
Debug
Breakpoints Variables Registers Modules
Mico32Build LEDTest.c LEDTest.c
Outline Disassembly
Console Tasks Memory

```
LEDtest
  Binaries
  Archives
  Debug
    Debug
    LEDtest.elf - [nonebe]
    LEDTest.o - [nonebe]
    LEDTest.d
    drivers.mk
    force_clean_archive_objs
    makefile
    subdir.mk
  Mico32Build
  LEDTest.c
  LEDTest.txt
  user.pref
```

```
LEDtest (1) [mico32 instruction set simulator]
  com.lattice.mdk.debug.mico32debugger (23.02.09 16:04) (Suspended)
    Thread [0] (Suspended: Breakpoint hit.)
      1 main() \cygdrive\d\Data\EmbeddedPresento\Mico32\Build\LEDtest\LEDTest.c:42
  Debugger Process (23.02.09 16:04)
  D:\Data\EmbeddedPresento\Mico32\Build\LEDtest\Debug\LEDtest.elf (23.02.09 16:04)
```

Name	Value
r24	0
r25	0
gp	33602256
fp	34602996
sp	34602972
ra	33555088
ea	0
ba	0
PC	33555080

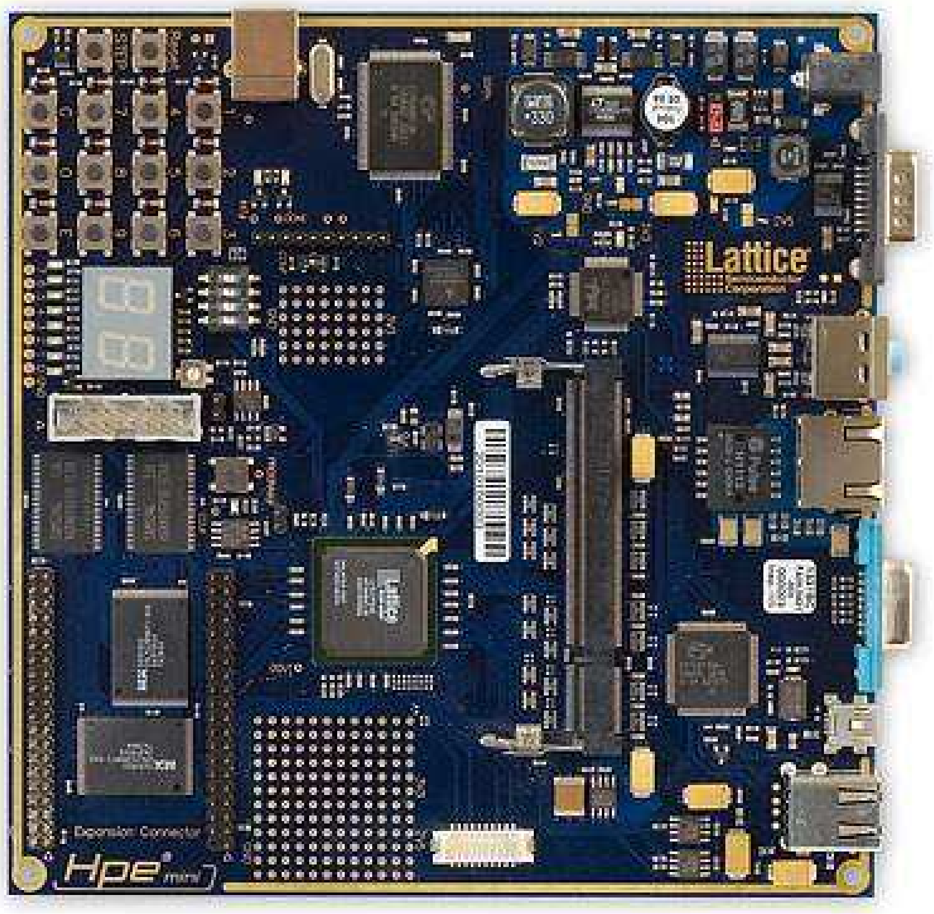
```
    /* scroll the LEDs, every 100 msecs forever */
    while(1){
        *((volatile unsigned int *)(leds->base)) = ~iValue;
        MicoSleepMilliSecs(100);
        if(iShiftLeft == 1){
            iValue = iValue << 1;
            if(iValue == 0x100){
                iValue = 0x40;
                iShiftLeft = 0;
            }
        }else{
            iValue = iValue >> 1;
```

```
0x02000280 <main+148>: not    r1,r1
0x02000284 <main+152>: sw     (r2+0),r
            MicoSleepMilliSecs(100);
0x02000288 <main+156>: mvi    r1,100
0x0200028c <main+160>: calli  0x200283
            if(iShiftLeft == 1){
0x02000290 <main+164>: lw     r1,(fp+-
0x02000294 <main+168>: cmpnei r1,r1,1
0x02000298 <main+172>: bne    r1,r0,0x
                iValue = iValue << 1;
0x0200029c <main+176>: lw     r1,(fp+0
0x020002a0 <main+180>: sli    r1,r1,1
```

```
LEDtest (1) [mico32 instruction set simulator] D:\Data\EmbeddedPresento\Mico32\Build\LEDtest\Debug\LEDtest.elf (23.02.09 16:04)
write to 0x80000080 length 4
```

Mico32 Evaluation Board for fast Tryout & Turnaround

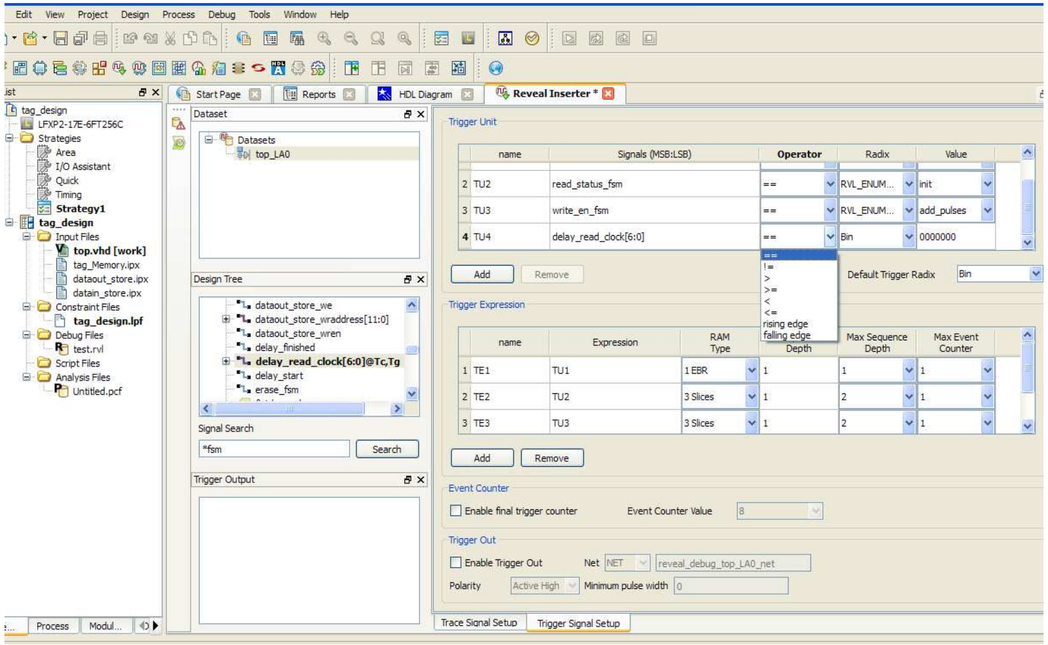
- LatticeMico32 System
- Development Board
  - LatticeECP2 available
  - DDR SODIMM socket
  - 2x128 Mbit Flash + 2x4 Mbit SRAM
  - USB 2.0 connector for programming
  - Flywire connector for programming
  - 9-pin RS232 serial port
  - 15-pin VGA connector for 64 colors
  - Ethernet 10/100 M full/half duplex
  - Multiple USB connectors
  - Sigma Delta D/A converter
  - Audio interface (line-in and line-out)
  - LCD connector for character displays
  - 25 MHz oscillator
  - Two-character 7-segment display



- Power Supply & USB Cable
- Also other evaluation boards available

Reveal - Debug Flow/Tool

- Uses Signal-Centric Method
  - Eliminates user need to “design” and connect debug core
- Two Tool Model
  - Inserter used to add debug and manage design
  - Analyzer used to connect / program / run / analyze device
- Powerful Complex Trigger Functions
  - =, !=, >, >=, <, =<, rising edge, falling edge
  - AND, OR, XOR, NOT, THEN, Next



Signal Search: *fsm Search

Trigger Unit

	name	Signals (MSB:LSB)	Operator	Radix	Value
2	TU2	read_status_fsm	==	RVL_ENUM...	init
3	TU3	write_en_fsm	==	RVL_ENUM...	add_pulses
4	TU4	delay_read_clock[6:0]	==	Bin	0000000

Default Trigger Radix: Bin

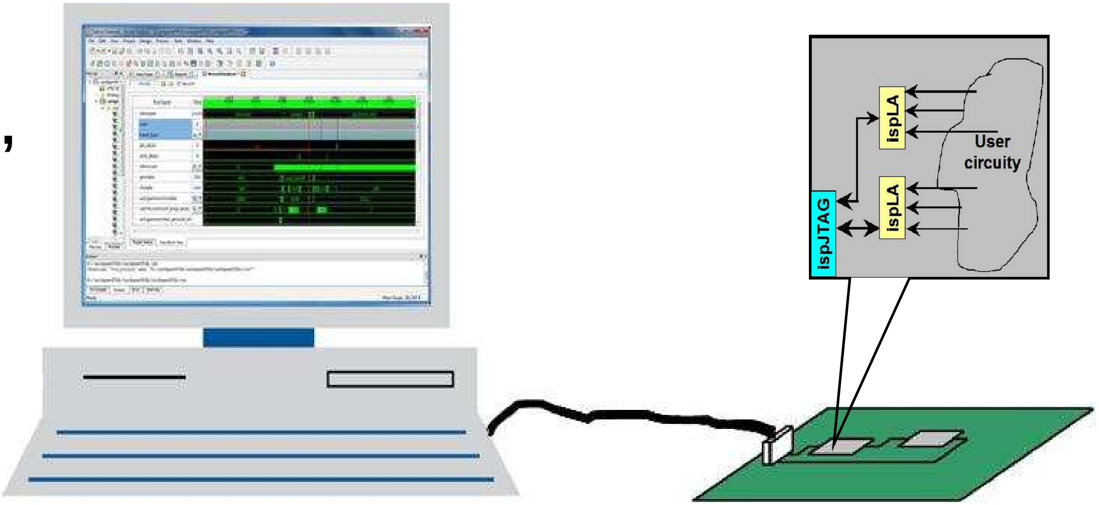
Trigger Expression

	name	Expression	RAM Type	Depth	Max Sequence Depth	Max Event Counter
1	TE1	TU1	1 EBR	1	1	1
2	TE2	TU2	3 Slices	1	2	1
3	TE3	TU3	3 Slices	1	2	1

☐ Enable final trigger counter Event Counter Value: 8

☐ Enable Trigger Out Net: NET reveal_debug_top_LA0_net

Polarity: Active High Minimum pulse width: 0

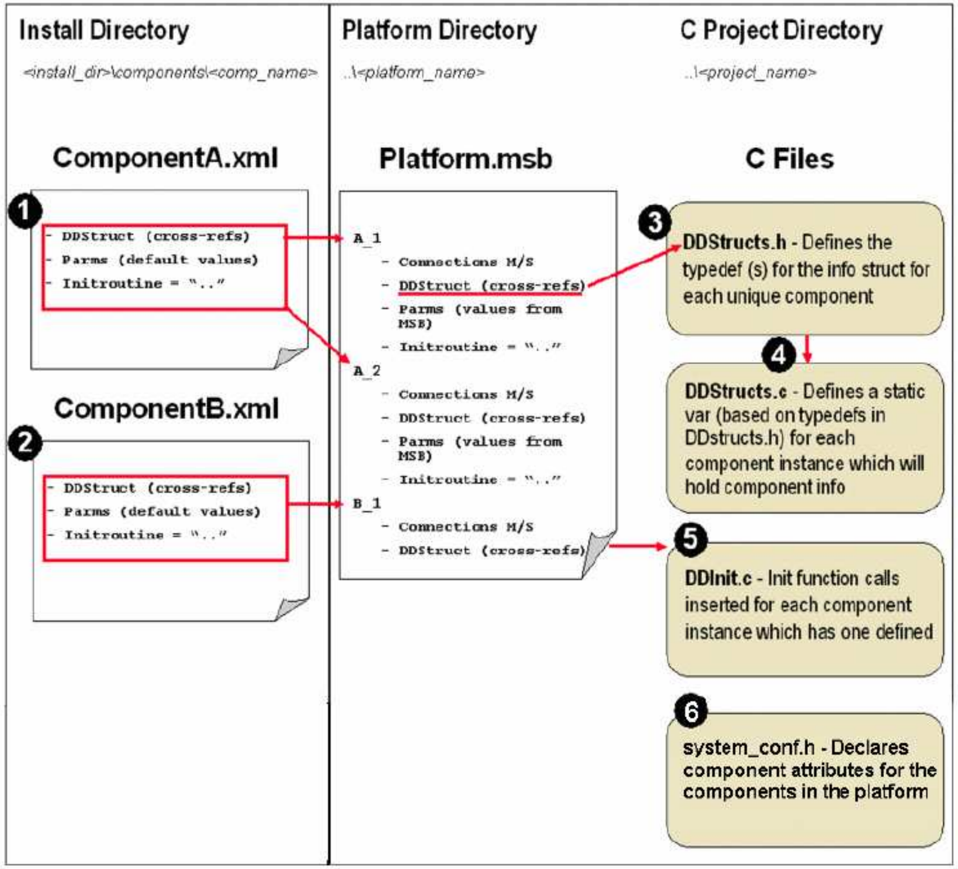


A bit on Software

- **SW development environment creates all the necessary drivers for accessing the selected components/peripherals.**
  - **Initialization routines**
  - **Access routines**
  - **Data/address/register structure**
  - **Call init from global init routine**
  - **Interrupt routine**

- **Build process creates:**
  - **DDStructs.c/.h**
  - **DDInit.c**
  - **system_conf.h**

- **Custom components will be supported by templates of the above.**



Install Directory

<install_dir>\components\<comp_name>

ComponentA.xml

1

```
- DDStruct (cross-refs)
- Parms (default values)
- Initroutine = ".."
```

ComponentB.xml

2

```
- DDStruct (cross-refs)
- Parms (default values)
- Initroutine = ".."
```

Platform Directory

..\<platform_name>

Platform.msb

```
A_1
    - Connections M/S
    - DDStruct (cross-refs)
    - Parms (values from
      MSB)
    - Initroutine = ".."
A_2
    - Connections M/S
    - DDStruct (cross-refs)
    - Parms (values from
      MSB)
    - Initroutine = ".."
B_1
    - Connections M/S
    - DDStruct (cross-refs)
```

C Project Directory

..\<project_name>

C Files

3 **DDStructs.h** - Defines the typedef (s) for the info struct for each unique component

4 **DDStructs.c** - Defines a static var (based on typedefs in DDstructs.h) for each component instance which will hold component info

5 **DDInit.c** - Init function calls inserted for each component instance which has one defined

6 system_conf.h - Declares component attributes for the components in the platform

AGENDA

Overview

Mico32 Embedded Processor

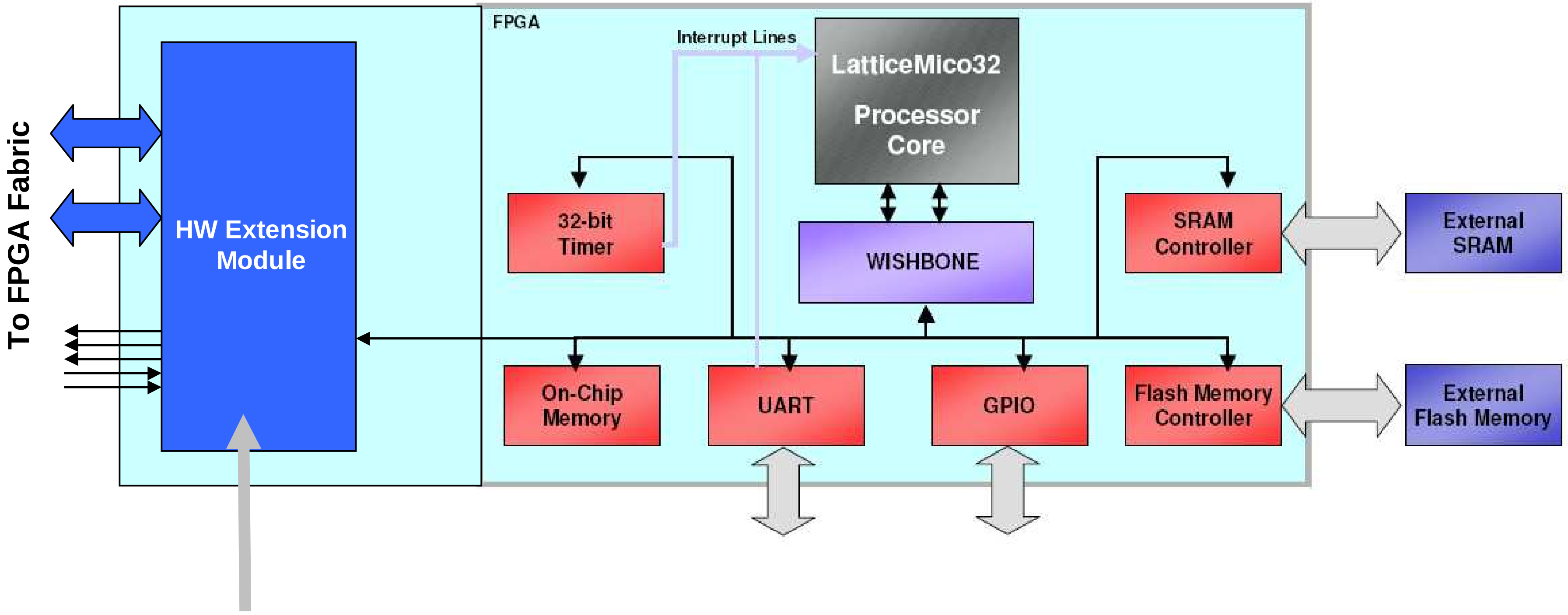
Development Tool Chain

HW/SW Example

Conclusion

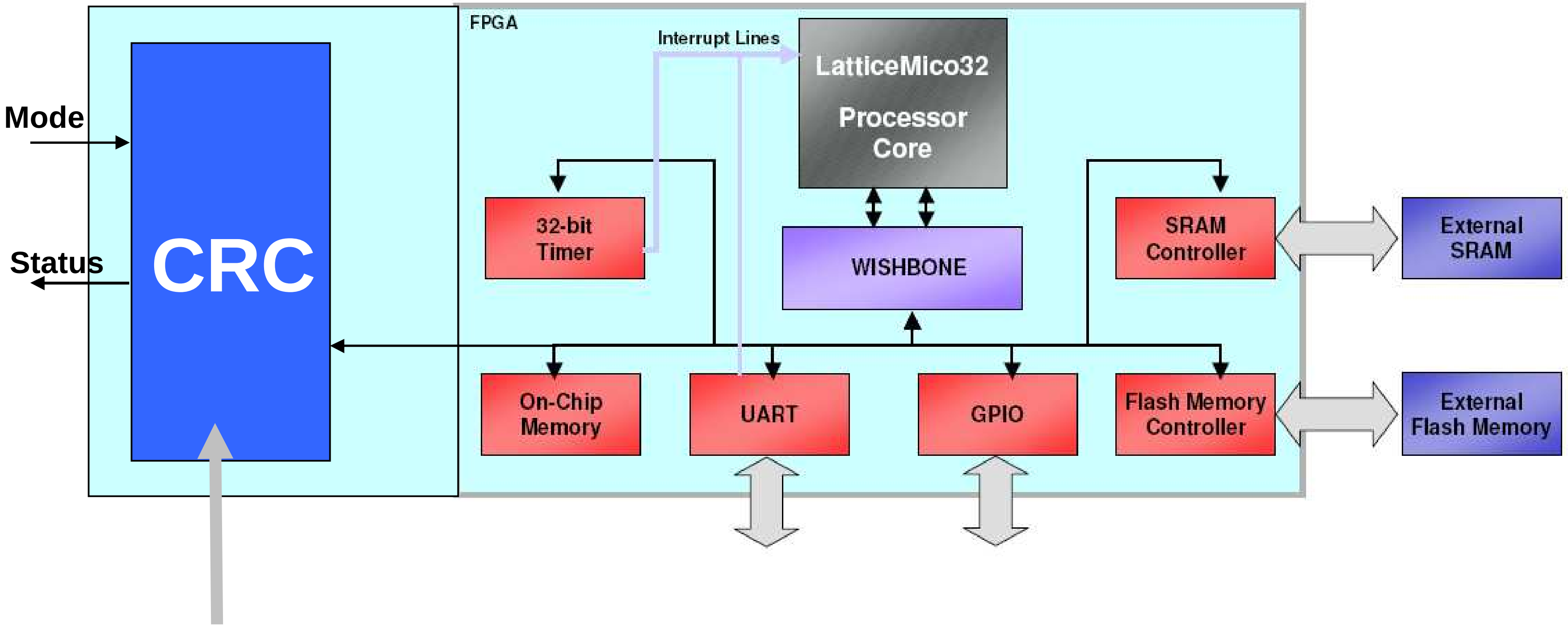
HW/SW EXAMPLE

Using an FPGA HW module as Custom Extension



Custom extension - can be any HW functionality, such as CRC calculation, framing/de-framing, extraction of control information, signal processing, data/memory transactions, …

EXAMPLE: CRC Calculation



Custom extension - can be any HW functionality, such as CRC calculation, framing/de-framing, extraction of control information, signal processing, data/memory transactions, …

CRC Generation with Custom Component

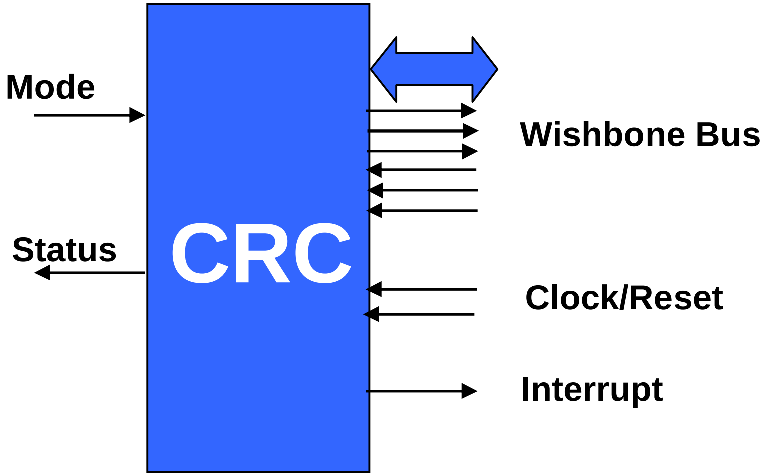
CRC Generation with Custom Component

Functionality

- Set initialization Value
- Write new data onto CRC
- Read current CRC
- Mode bits, Status bits, Interrupt

Registers

- 0x00: CRC Init Value
- 0x04: CRC Polynom
- 0x08: Write new data word
- 0x0C: Read CRC value
- 0x10: Interrupt Register
- 0x14: Set mode bits
- 0x18: Read status bits



Interface

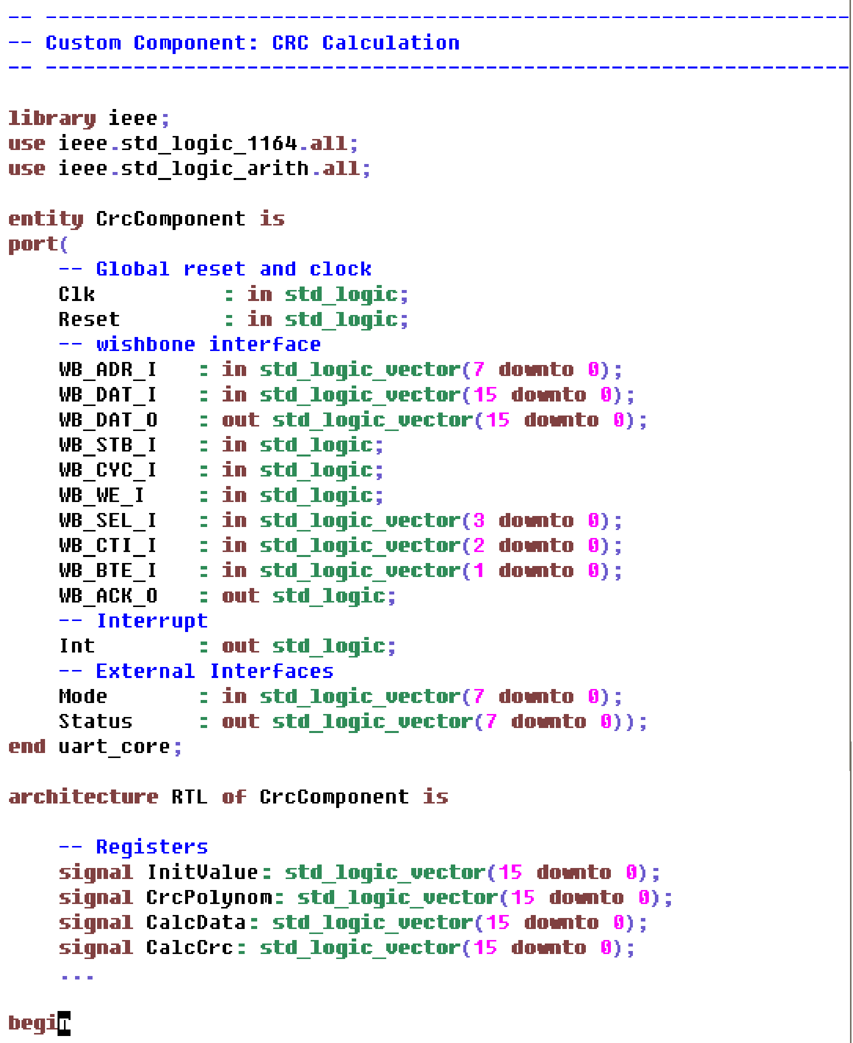
- Clock, Reset
- Interrupt
- Wichbone signals
- Mode/Status (external signals)

HW/SW EXAMPLE

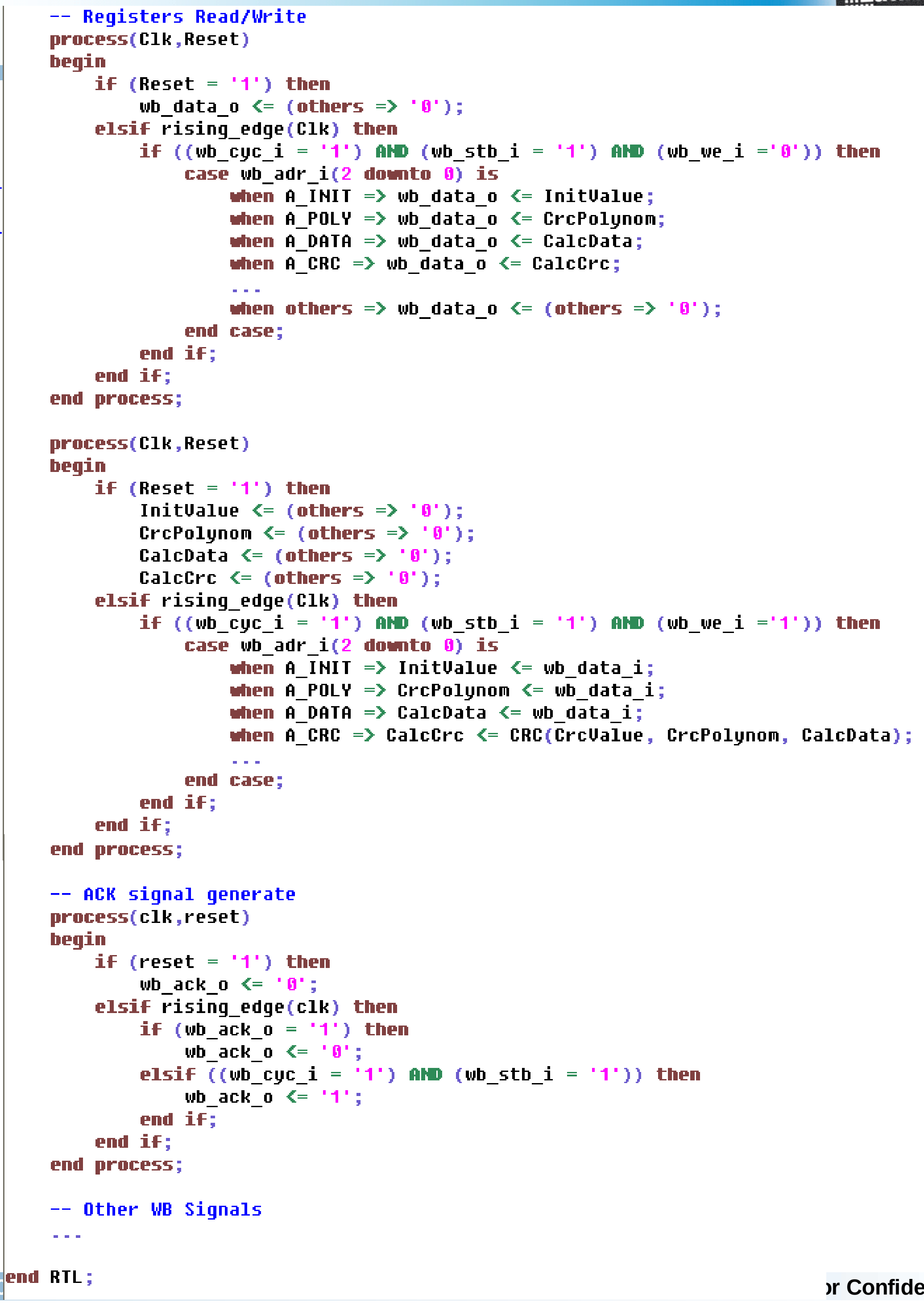
How to Define and Implement a Custom Component

- Definition of the Custom Component (VHDL/Verilog)
- Assign Wishbone signals of the custom component to the Wishbone signals used by Mico32 for internal communications.
- Definition of external Inputs/Outputs
- Assign design files for HW implementation of the Custom Component.
- Definition of the parameters, such as address range, base address, data width, interrupts, ...
  - This parameters will then be used during configuration of the custom component
- SW Template for Definition of the used data types
- SW Files Template for access routines, init routines, any other SW aspects to be assigned to the custom components
  - This can be used functionally in the created SW framework
  - Init routines will be included automatically into the SW initialization mechanism

Define Custom Component (VHDL/Verilog)

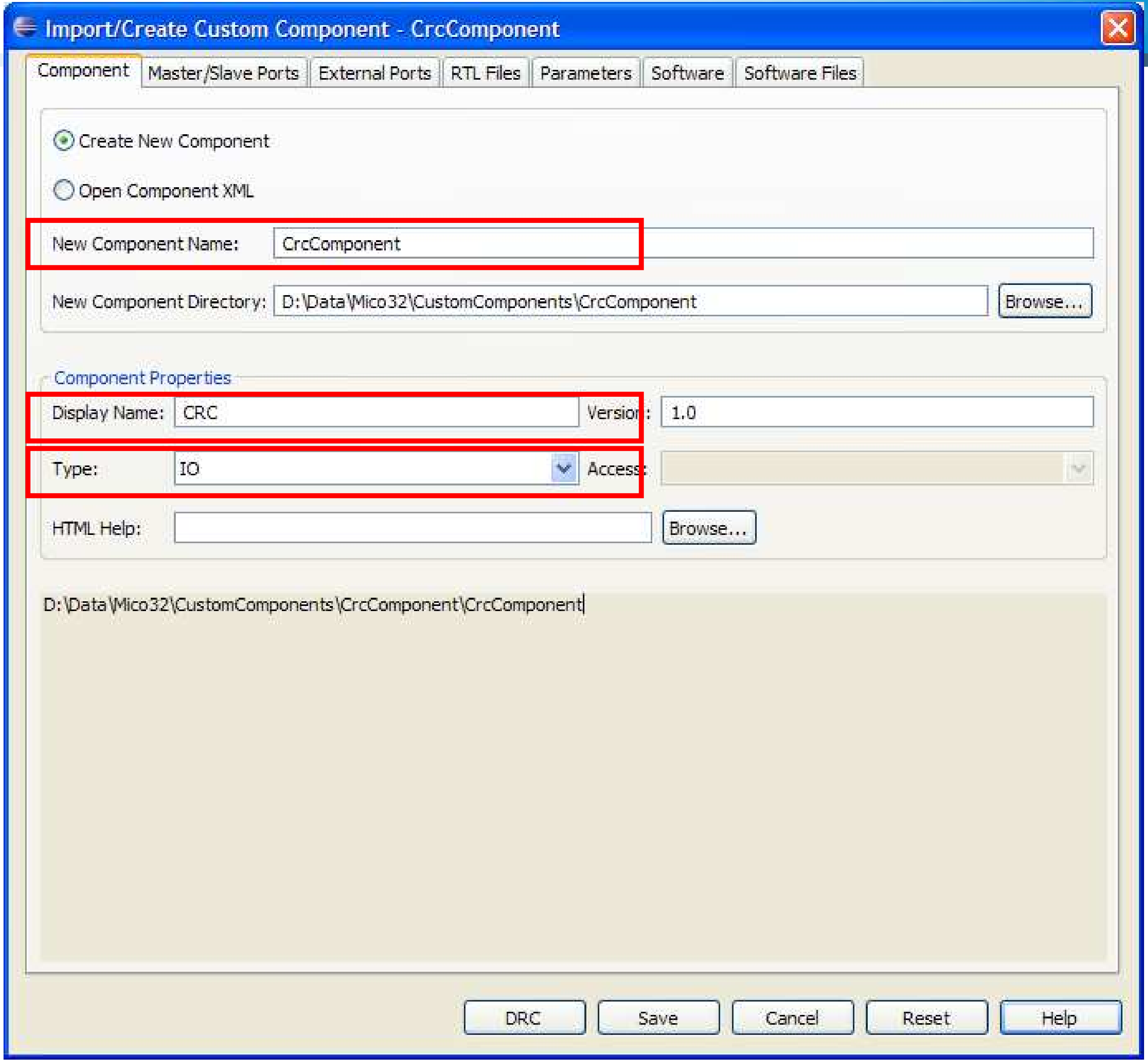


```
-- ------------------------------------------------------------------
-- Custom Component: CRC Calculation
-- ------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity CrcComponent is
port(
    -- Global reset and clock
    Clk           : in std_logic;
    Reset         : in std_logic;
    -- wishbone interface
    WB_ADR_I    : in std_logic_vector(7 downto 0);
    WB_DAT_I    : in std_logic_vector(15 downto 0);
    WB_DAT_O    : out std_logic_vector(15 downto 0);
    WB_STB_I    : in std_logic;
    WB_CYC_I    : in std_logic;
    WB_WE_I     : in std_logic;
    WB_SEL_I    : in std_logic_vector(3 downto 0);
    WB_CTI_I    : in std_logic_vector(2 downto 0);
    WB_BTE_I    : in std_logic_vector(1 downto 0);
    WB_ACK_O    : out std_logic;
    -- Interrupt
    Int         : out std_logic;
    -- External Interfaces
    Mode        : in std_logic_vector(7 downto 0);
    Status      : out std_logic_vector(7 downto 0));
end uart_core;

architecture RTL of CrcComponent is

    -- Registers
    signal InitValue: std_logic_vector(15 downto 0);
    signal CrcPolynom: std_logic_vector(15 downto 0);
    signal CalcData: std_logic_vector(15 downto 0);
    signal CalcCrc: std_logic_vector(15 downto 0);
    ...

begin
```



```
-- Registers Read/Write
process(Clk,Reset)
begin
    if (Reset = '1') then
        wb_data_o <= (others => '0');
    elsif rising_edge(Clk) then
        if ((wb_cyc_i = '1') AND (wb_stb_i = '1') AND (wb_we_i ='0')) then
            case wb_adr_i(2 downto 0) is
                when A_INIT => wb_data_o <= InitValue;
                when A_POLY => wb_data_o <= CrcPolynom;
                when A_DATA => wb_data_o <= CalcData;
                when A_CRC => wb_data_o <= CalcCrc;
                ...
                when others => wb_data_o <= (others => '0');
            end case;
        end if;
    end if;
end process;

process(Clk,Reset)
begin
    if (Reset = '1') then
        InitValue <= (others => '0');
        CrcPolynom <= (others => '0');
        CalcData <= (others => '0');
        CalcCrc <= (others => '0');
    elsif rising_edge(Clk) then
        if ((wb_cyc_i = '1') AND (wb_stb_i = '1') AND (wb_we_i ='1')) then
            case wb_adr_i(2 downto 0) is
                when A_INIT => InitValue <= wb_data_i;
                when A_POLY => CrcPolynom <= wb_data_i;
                when A_DATA => CalcData <= wb_data_i;
                when A_CRC => CalcCrc <= CRC(CrcValue, CrcPolynom, CalcData);
                ...
            end case;
        end if;
    end if;
end process;

-- ACK signal generate
process(clk,reset)
begin
    if (reset = '1') then
        wb_ack_o <= '0';
    elsif rising_edge(clk) then
        if (wb_ack_o = '1') then
            wb_ack_o <= '0';
        elsif ((wb_cyc_i = '1') AND (wb_stb_i = '1')) then
            wb_ack_o <= '1';
        end if;
    end if;
end process;

-- Other WB Signals
...

end RTL;
```

Define Component



Import/Create Custom Component - CrcComponent

● Create New Component

○ Open Component XML

New Component Name: CrcComponent

New Component Directory: D:\Data\Mico32\CustomComponents\CrcComponent

Component Properties

Display Name: CRC

Version: 1.0

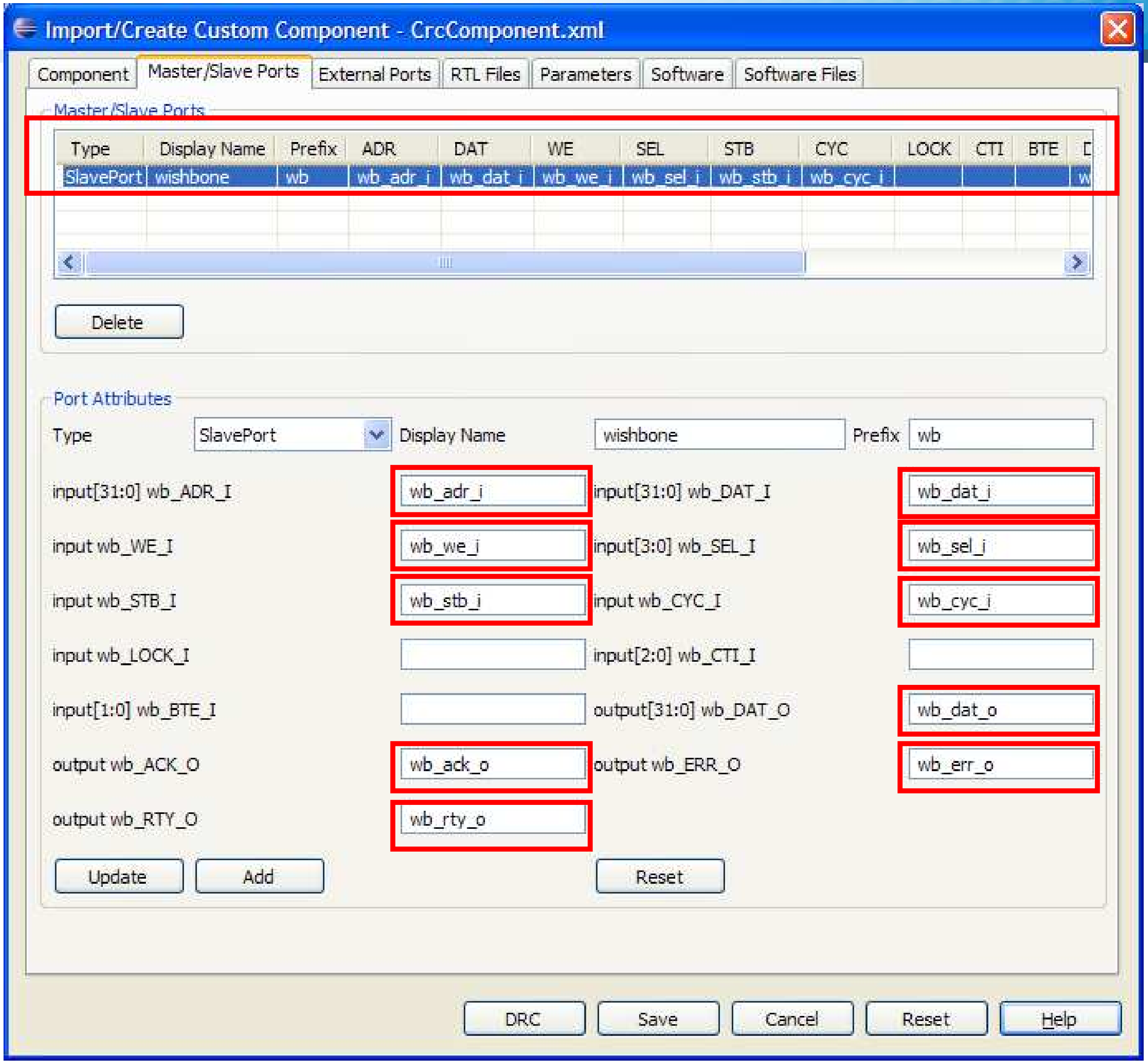
Type: IO

Access:

HTML Help:

D:\Data\Mico32\CustomComponents\CrcComponent\CrcComponent

Assign Wishbone Interface



Import/Create Custom Component - CrcComponent.xml

Master/Slave Ports

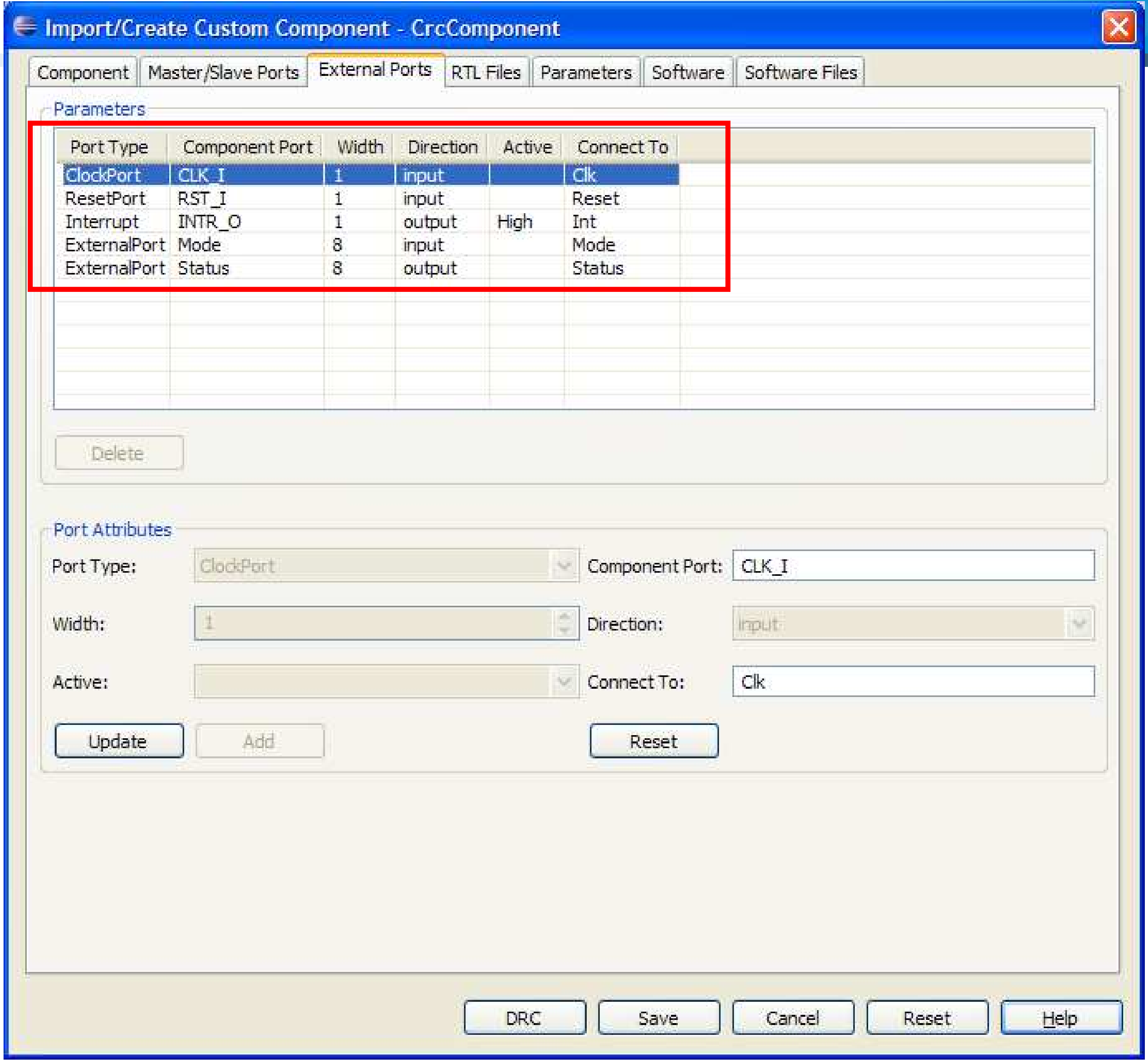
Type	Display Name	Prefix	ADR	DAT	WE	SEL	STB	CYC	LOCK	CTI	BTE	D
SlavePort	wishbone	wb	wb_adr_i	wb_dat_i	wb_we_i	wb_sel_i	wb_stb_i	wb_cyc_i				w

Port Attributes

Type: SlavePort
Display Name: wishbone
Prefix: wb

input[31:0] wb_ADR_I: wb_adr_i
input[31:0] wb_DAT_I: wb_dat_i
input wb_WE_I: wb_we_i
input[3:0] wb_SEL_I: wb_sel_i
input wb_STB_I: wb_stb_i
input wb_CYC_I: wb_cyc_i
input wb_LOCK_I:
input[2:0] wb_CTI_I:
input[1:0] wb_BTE_I:
output[31:0] wb_DAT_O: wb_dat_o
output wb_ACK_O: wb_ack_o
output wb_ERR_O: wb_err_o
output wb_RTY_O: wb_rty_o

External Ports



Import/Create Custom Component - CrcComponent

Parameters

Port Type	Component Port	Width	Direction	Active	Connect To
ClockPort	CLK_I	1	input		Clk
ResetPort	RST_I	1	input		Reset
Interrupt	INTR_O	1	output	High	Int
ExternalPort	Mode	8	input		Mode
ExternalPort	Status	8	output		Status

Port Attributes

Port Type: ClockPort

Component Port: CLK_I

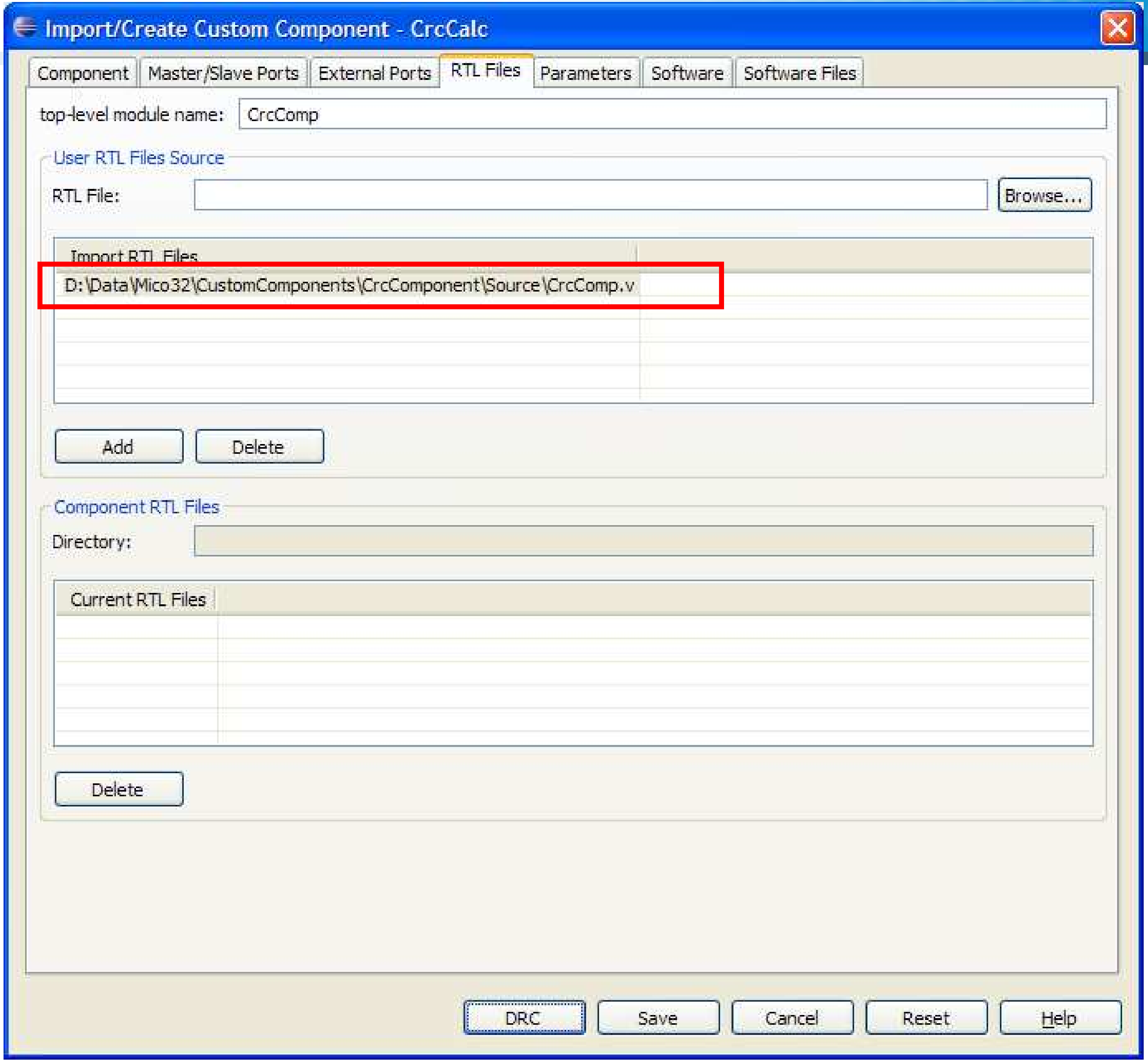
Width: 1

Direction: input

Active:

Connect To: Clk

RTL Files



Import/Create Custom Component - CrcCalc

top-level module name: CrcComp

User RTL Files Source

RTL File:

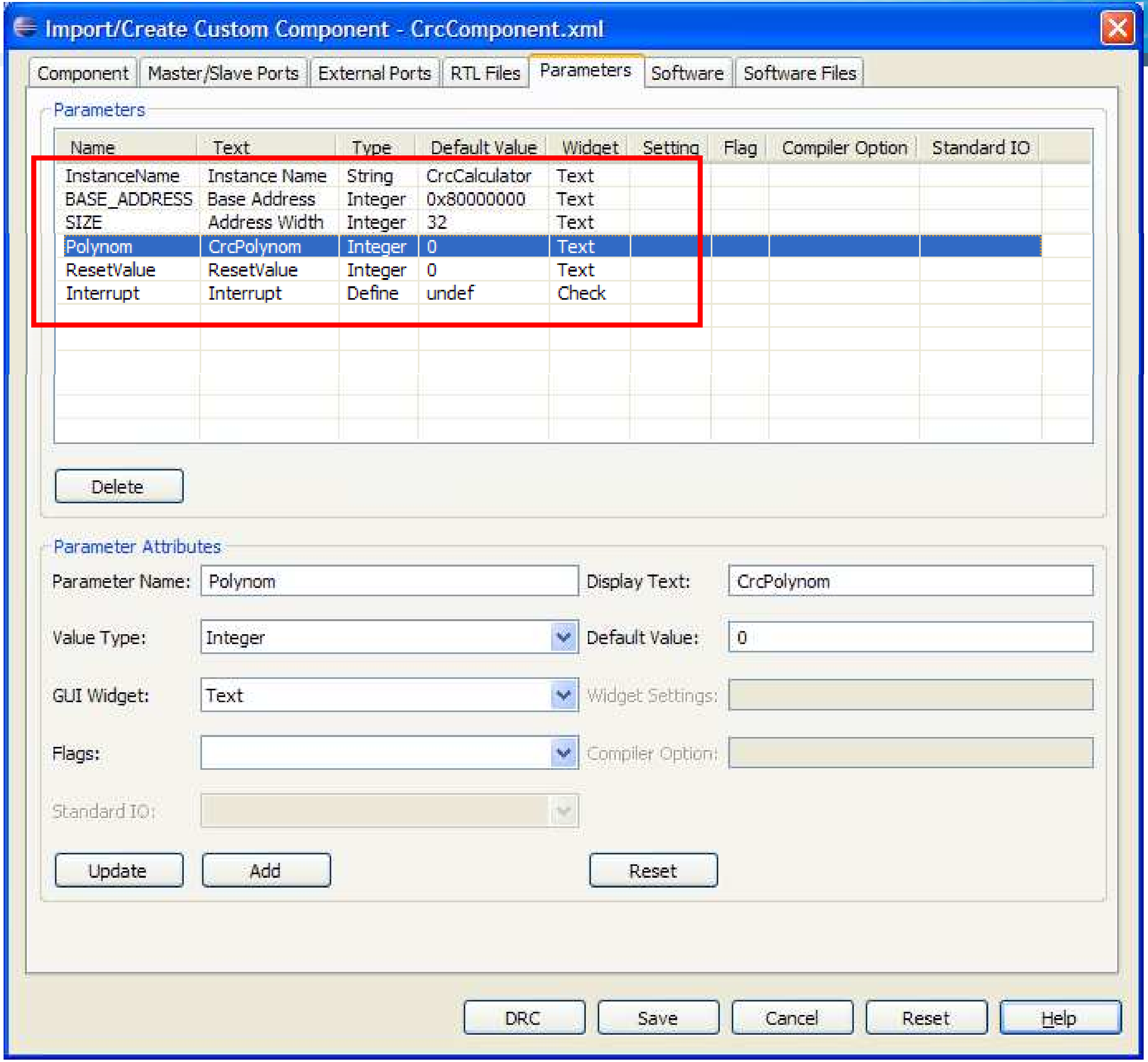
Import RTL Files
D:\Data\Mico32\CustomComponents\CrcComponent\Source\CrcComp.v

Component RTL Files

Directory:

Current RTL Files

Parameters



Parameters

Name	Text	Type	Default Value	Widget	Setting	Flag	Compiler Option	Standard IO
InstanceName	Instance Name	String	CrcCalculator	Text				
BASE_ADDRESS	Base Address	Integer	0x80000000	Text				
SIZE	Address Width	Integer	32	Text				
Polynom	CrcPolynom	Integer	0	Text				
ResetValue	ResetValue	Integer	0	Text				
Interrupt	Interrupt	Define	undef	Check				

Parameter Attributes

Parameter Name: Polynom

Display Text: CrcPolynom

Value Type: Integer

Default Value: 0

GUI Widget: Text

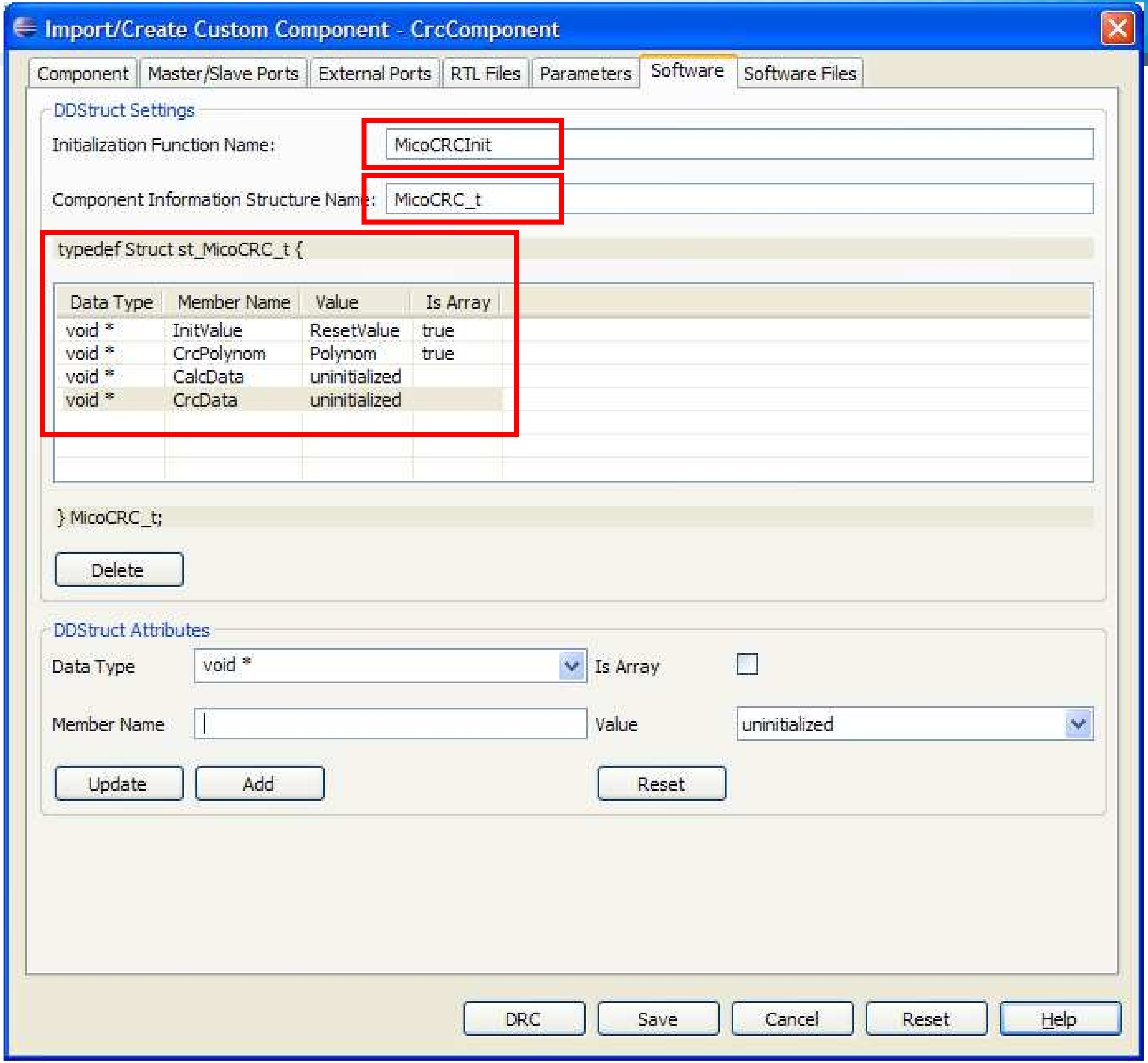
Widget Settings:

Flags:

Compiler Option:

Standard IO:

Define SW Data Type



DDStruct Settings

Initialization Function Name: MicoCRCInit

Component Information Structure Name: MicoCRC_t

typedef Struct st_MicoCRC_t {

Data Type	Member Name	Value	Is Array
void *	InitValue	ResetValue	true
void *	CrcPolynom	Polynom	true
void *	CalcData	uninitialized	
void *	CrcData	uninitialized	

} MicoCRC_t;

DDStruct Attributes

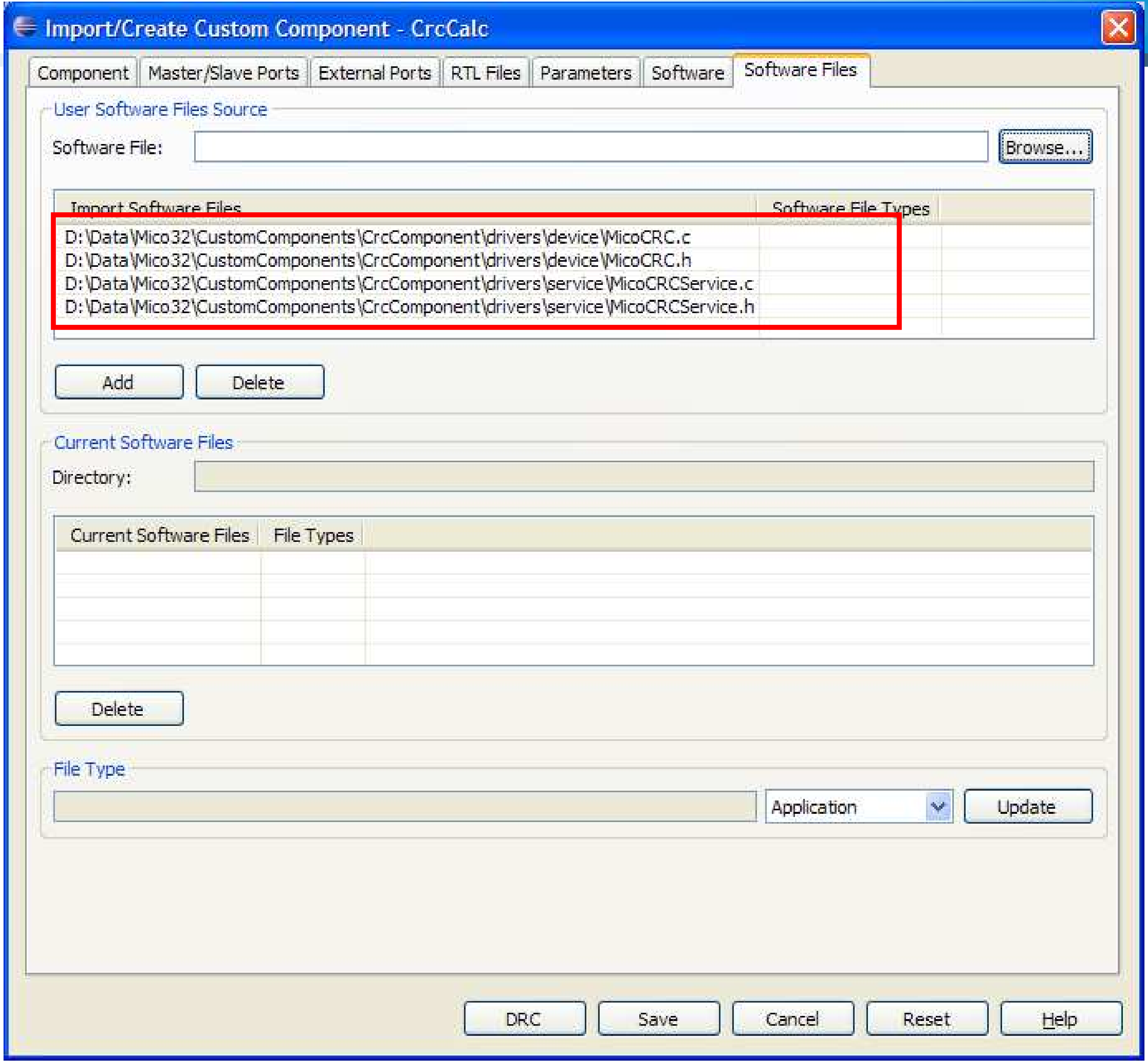
Data Type: void *

Is Array: □

Member Name:

Value: uninitialized

Assign SW Files



Software File:

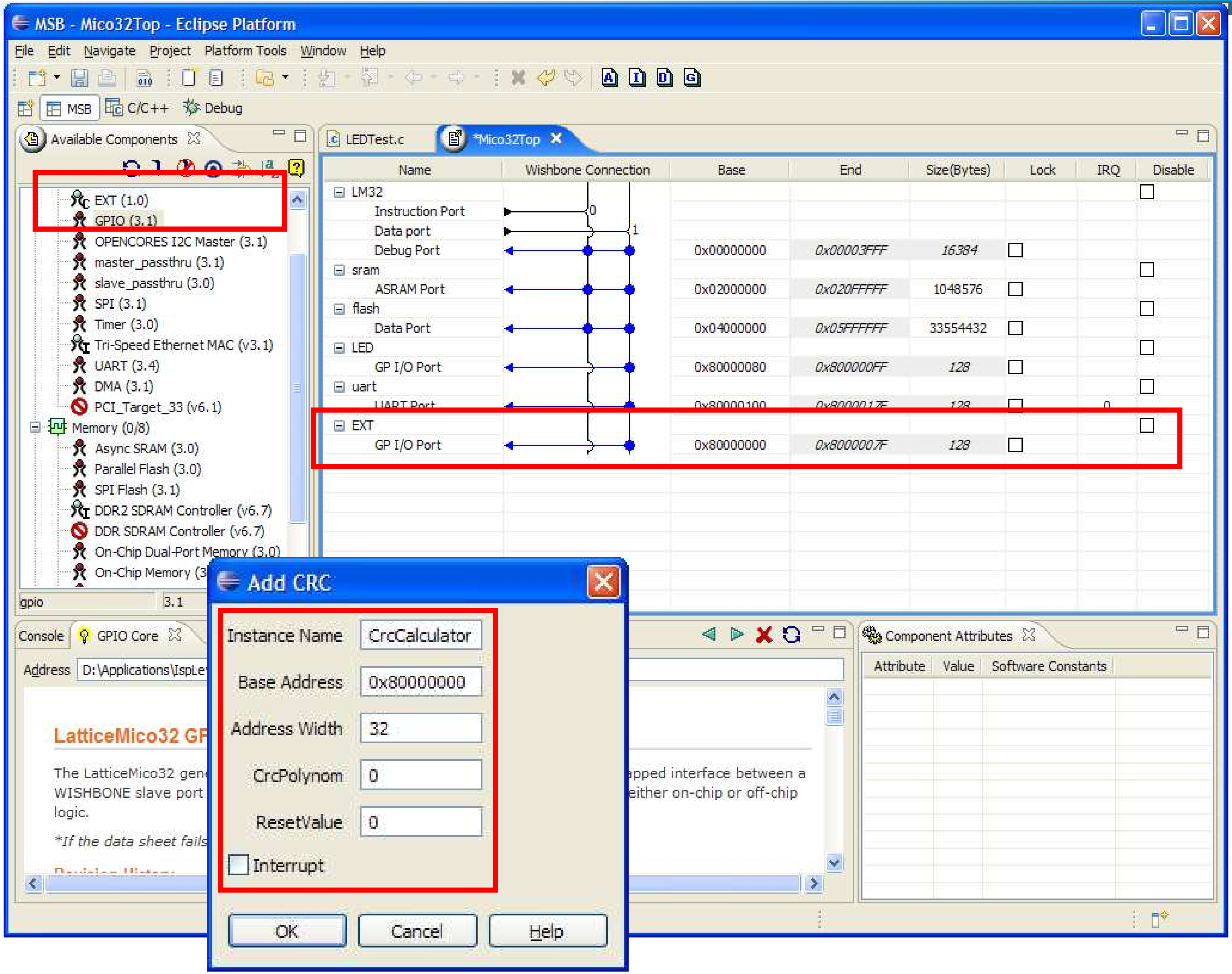
Import Software Files	Software File Types
D:\Data\Mico32\CustomComponents\CrcComponent\drivers\device\MicoCRC.c	
D:\Data\Mico32\CustomComponents\CrcComponent\drivers\device\MicoCRC.h	
D:\Data\Mico32\CustomComponents\CrcComponent\drivers\service\MicoCRCService.c	
D:\Data\Mico32\CustomComponents\CrcComponent\drivers\service\MicoCRCService.h	

Directory:

Current Software Files	File Types

File Type: Application

Include Custom Component into Your Design



Name	Wishbone Connection	Base	End	Size(Bytes)	Lock	IRQ	Disable
⊟ LM32							□
Instruction Port	0						
Data port	1						
Debug Port		0x00000000	0x00003FFF	16384	□		
⊟ sram							□
ASRAM Port		0x02000000	0x020FFFFF	1048576	□		
⊟ flash							□
Data Port		0x04000000	0x05FFFFFF	33554432	□		
⊟ LED							□
GP I/O Port		0x80000080	0x800000FF	128	□		
⊟ uart							□
UART Port		0x80000100	0x8000017F	128	□	0	
⊟ EXT							□
GP I/O Port		0x80000000	0x8000007F	128	□		

Add CRC

Instance Name	CrcCalculator
Base Address	0x80000000
Address Width	32
CrcPolynom	0
ResetValue	0

□ Interrupt

Custom Component Header File

```
/*------------------*/
/* MicoCRC.h */
/*------------------*/

/*********************************************************
**
**  Name: MicoCRC.h
**
**  Description:
**       Declares CRC register structure and
**       macros/functions for manipulating CRC
**
*********************************************************

    /*
     *****************************************
              CRC REGISTER MAPPING
     *****************************************
     */
    typedef struct st_MicoCRC_t{
        volatile unsigned int InitValue     /* R/W:
        volatile unsigned int CrcPolynom    /* R/W:
        volatile unsigned int CalcData      /* R/W:
        volatile unsigned int CalcCrc       /* R/W:
        volatile unsigned int Mode          /* R/W:
        volatile unsigned int Status        /* R/W:
        ...
    }MicoCRC_t;
```

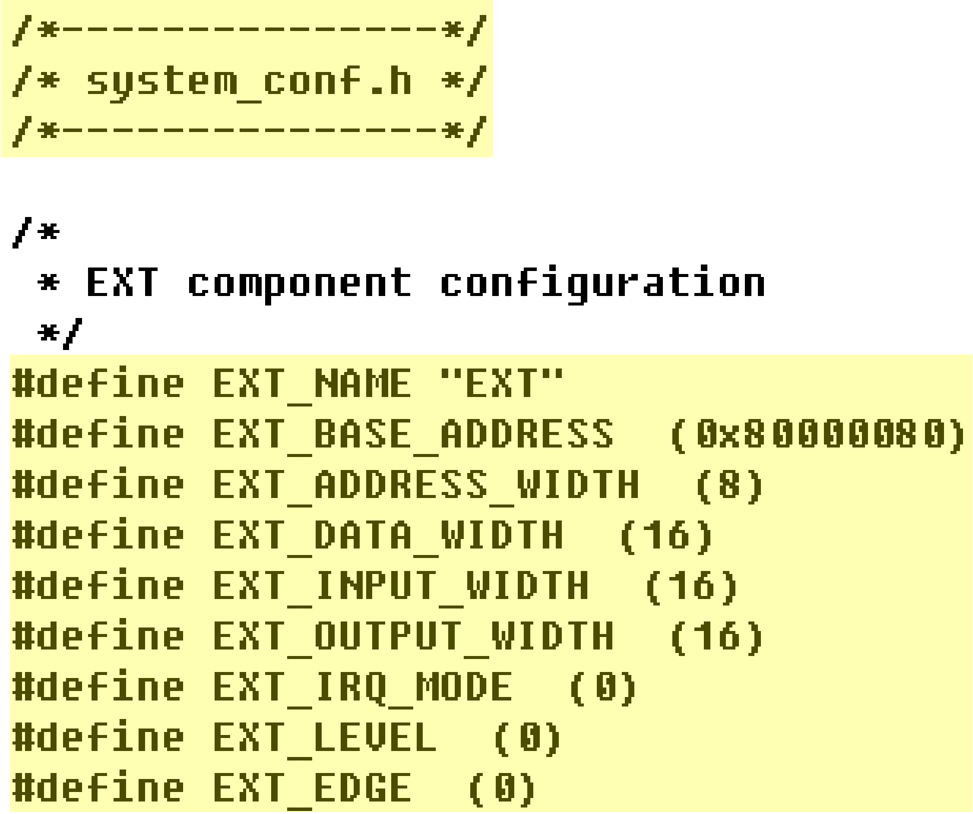
```
    /*
     ********************************************
         MACROS FOR ACCESSING CRC REGISTERS
     ********************************************
     */

    /* reads data register */
    #define MICO_CRC_READ_DATA(X,Y)    \
        (Y)=((volatile MicoCRC_t *)((X)->base))->data

    /* writes data-register */
    #define MICO_CRC_WRITE_DATA(X,Y)   \
        ((volatile MicoCRC_t *)((X)->base))->data=(Y)

    /* reads irq-mask register */
    #define MICO_CRC_CALC_CRC(X,Y)   \
        (Y) = ((volatile MicoCRC_t *)((X)->base))->irqMask

    /* writes irq-mask register */
    #define MICO_CRC_RESET(X,Y)   \
        ((volatile MicoCRC_t *)((X)->base))->irqMask = (Y)

    ...

/**************************************************************
 * functions
 **************************************************************

/* initializes Mico32 CRC peripheral */
void MicoCRCInit( MicoCRCCtx_t *ctx );
```

SW generated Files: DDStruct, DDInit

```c
/*-------------*/
/* DDStructs.h */
/*-------------*/

/*Device-driver structure for extension*/
#define MicoCRCCtx_t_DEFINED (1)
typedef struct st_MicoCRCCtx_t {
    const char*    InstancecName;
    unsigned int   BaseAddress;
    unsigned int   intrLevel;
    unsigned int   CrcValue;
    unsigned int   CrcPolynom;
    unsigned int   data_width;
    unsigned int   input_width;
    unsigned int   output_width;
    unsigned int   intr_enable;
} MicoCRCCtx_t;

/* extension instance CRC*/
extern struct st_MicoCRCCtx_t extension_CRC;

/* declare CRC instance of extension */
extern void MicoCRCInit(struct st_MicoCRCCtx_t*);
```

```c
/*----------*/
/* DDInit.c */
/*----------*/

void LatticeDDInit(void)
{
    /* initialize LM32 instance of lm32_top */
    LatticeMico32Init(&lm32_top_LM32);

    /* initialize flash instance of asram_top */
    LatticeMico32InitCFIFlashDriver(&asram_top_flash);

    /* initialize CRC instance of extension */
    MicoCRCInit(&extension_CRC);

    /* initialize uart instance of uart_core */
    MicoUartInit(&uart_core_uart);

    /* invoke application's main routine*/
    main();
}
```

System Defines



```
/*---------------*/
/* system_conf.h */
/*---------------*/

/*
 * EXT component configuration
 */
#define EXT_NAME "EXT"
#define EXT_BASE_ADDRESS  (0x80000080)
#define EXT_ADDRESS_WIDTH  (8)
#define EXT_DATA_WIDTH  (16)
#define EXT_INPUT_WIDTH  (16)
#define EXT_OUTPUT_WIDTH  (16)
#define EXT_IRQ_MODE  (0)
#define EXT_LEVEL  (0)
#define EXT_EDGE  (0)
```

SW Build Process

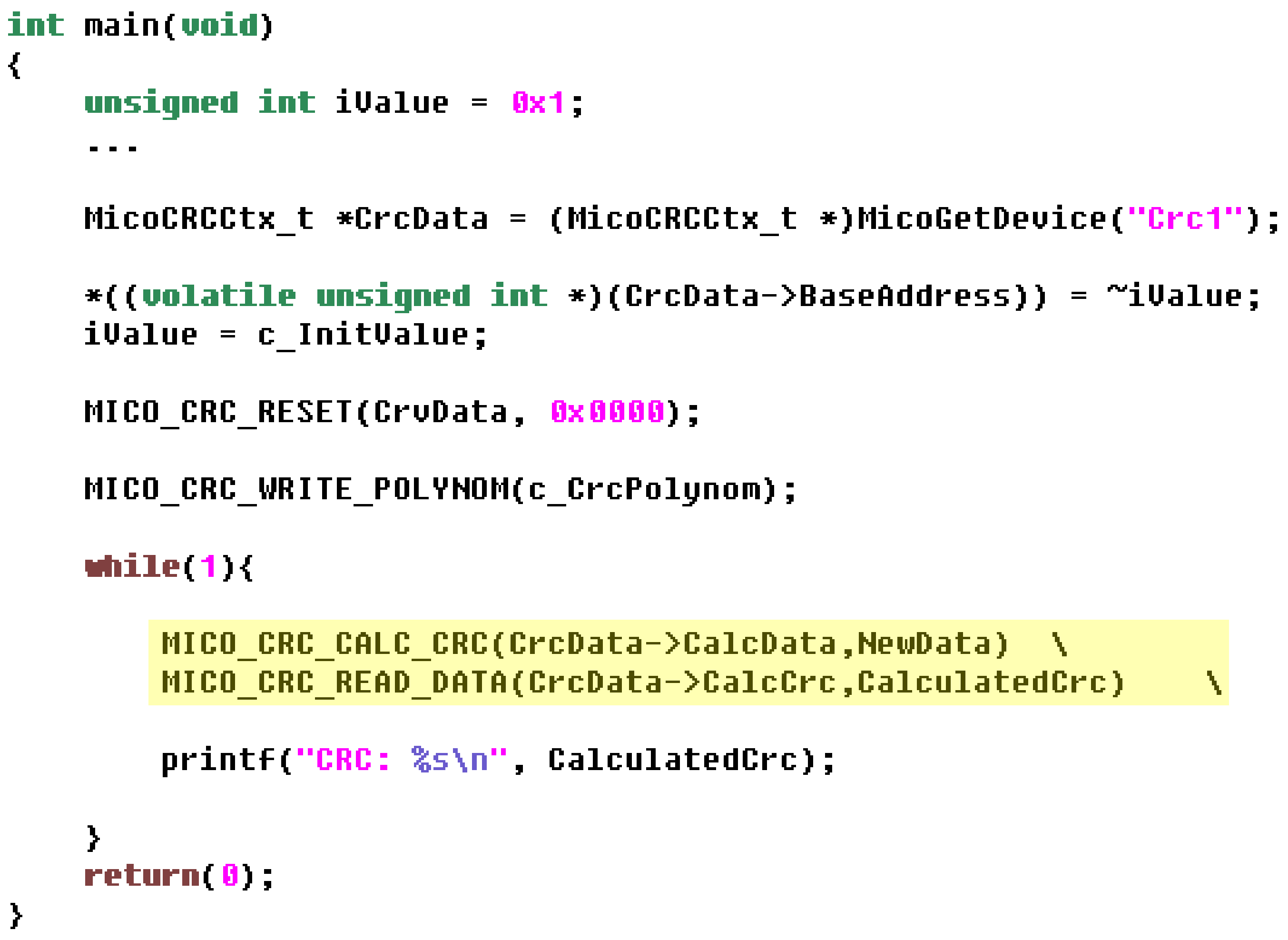
Build process creates:

- DDStructs.c/DDStructs.h:
  Contains data types & device driver
- DDInit.c:
  Initialization routines for Mico32 & peripheral modules
- system_conf.h:
  Configuration data such as address range, base address, ...

Application SW (main) can now access to this custom component via these access routines & defines

Init routines will be included automatically into the SW initialization mechanism

SW Usage Model



```
int main(void)
{
    unsigned int iValue = 0x1;
    ...

    MicoCRCCtx_t *CrcData = (MicoCRCCtx_t *)MicoGetDevice("Crc1");

    *((volatile unsigned int *)(CrcData->BaseAddress)) = ~iValue;
    iValue = c_InitValue;

    MICO_CRC_RESET(CrvData, 0x0000);

    MICO_CRC_WRITE_POLYNOM(c_CrcPolynom);

    while(1){

        MICO_CRC_CALC_CRC(CrcData->CalcData,NewData)  \
        MICO_CRC_READ_DATA(CrcData->CalcCrc,CalculatedCrc)    \

        printf("CRC: %s\n", CalculatedCrc);

    }
    return(0);
}
```

AGENDA

Overview

Mico32 Embedded Processor

Development Tool Chain

HW/SW Example

Conclusion

CONCLUSION

Implementation of embedded RISC processor combined with custom component allows simple & powerful interaction between SW programmable solution and HW implementations

Processor architecture created by MSB – Mico32 System Builder – graphical user interface by point & click

Fully supported by GUI for definition of Custom Component (incl. Assignment of SW drivers)

All relevant SW drivers automatically generated for standard peripherals and custom components

Ease of use leads to fast path to application programming

Free of Charge Download:
http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/mico32developmenttools.cfm

Thank you …