IP Generators – The next wave of design creation

Neena Chandawale (Host)

Amanjyot Kaur (Presenter)

Agenda

- Introduction to IP generators
- Challenges in IP reuse
- Solutions
  - Chisel based
  - IDesignSpec™ – Register automation
  - ISequenceSpec™ – Custom sequence automation
  - SLIP-G – Standard Library of IPs Generator
- SLIP-G Overview
  - Configurability
  - Customizability
  - Standard sequences
- Deep dive into SLIP-G
  - GPIO
  - I2C
  - TIMER
  - PIC
- Benefits of IP generators
- Conclusion

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Introduction

- Progression of design methodologies
- Design abstraction
- IP generators

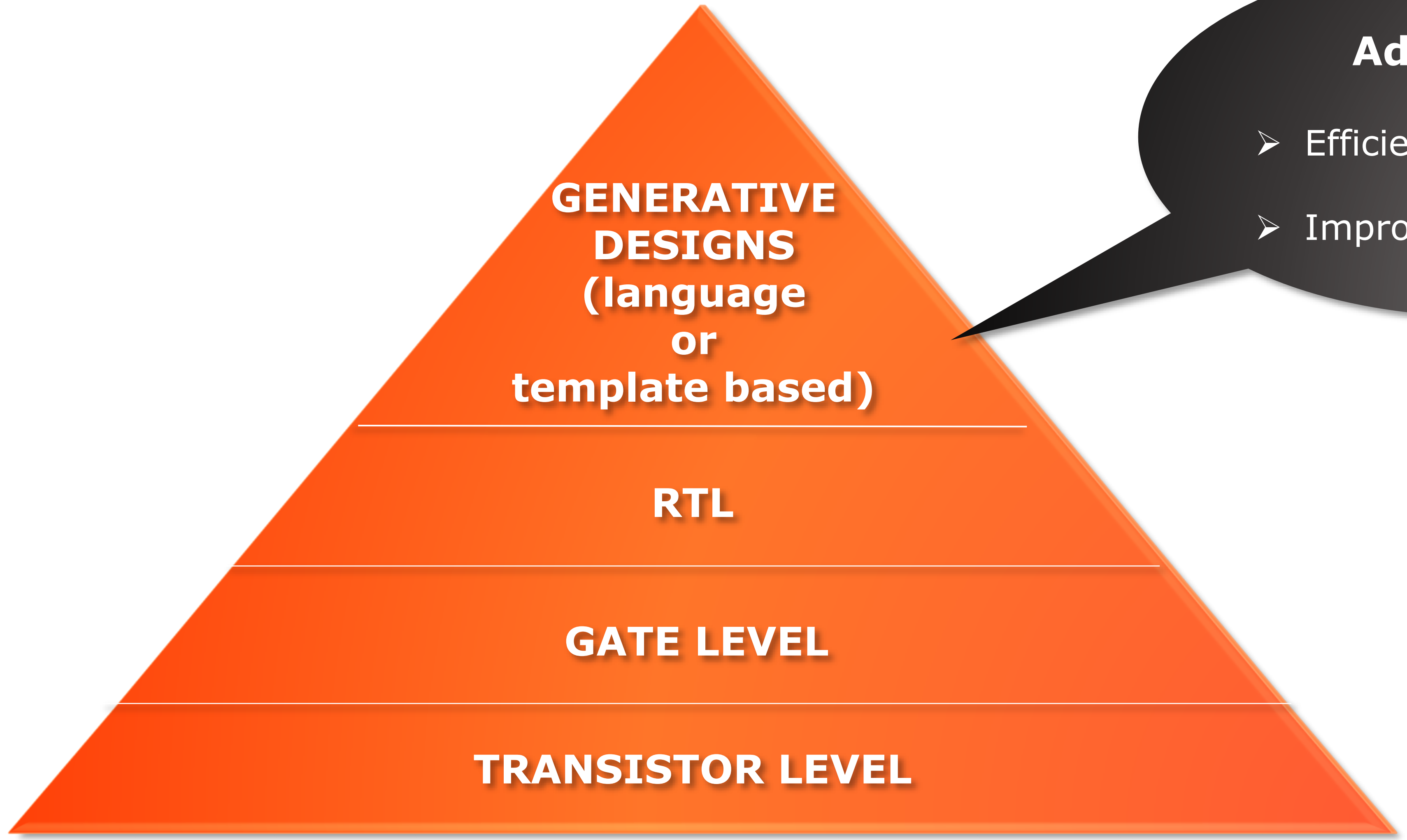
Challenges in IP reuse

Progression of design methodologies

- Designs evolved from hundreds of transistors to hundreds of billions over last several decades
- Shrinking transistor and increasing transistor count drove various design and verification methodologies
- Abstraction is the key to managing ever-increasing complexity and scale of ASIC/SoC designs

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Design abstraction



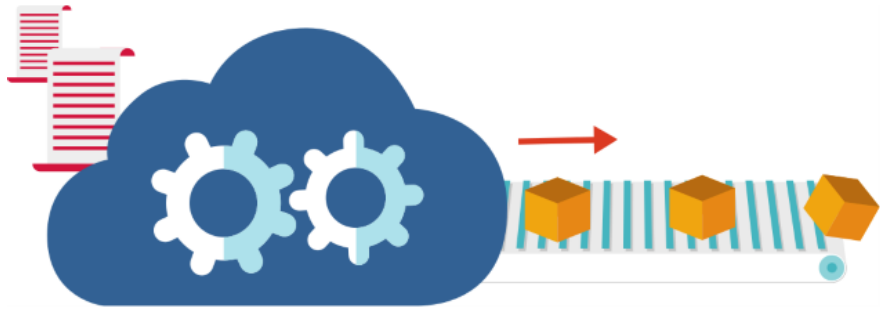
Advantages

- Efficient design flows
- Improved productivity

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Challenges in IP reuse

- RTL for an IP can be reused but it is quite brittle
  - Breaks when reused in a substantially changed environment
- Parameters based configuration is typically used to keep the IP flexible for reuse
  - Parameters are not enough
    - Changes in the port-list cannot be handled by parameters
- Customizations for the IP is not possible once the RTL IP is created
  - Example: Adding a custom field in an existing IP’s register
- It’s not just about the RTL
  - There are other aspects of the development flow that are impacted
    - Example: Firmware, Software
- **Solution**: Generate design IPs including RTL, firmware access code



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Introduction to IP generators

Must-have features

- Ease of generation with minimal time and effort
- Generated code should not be encrypted and indistinguishable from hand-crafted code
- Should provide appropriate error messages
- Ability to customize the IPs
  - Example: add functionality such as additional fields and registers to IP's reg-map
- Ability to configure the IPs
  - Example: set values for the parameters
- IPs must be fully tested, verified and validated

Creation of correct-by-construction, reusable designs, faster

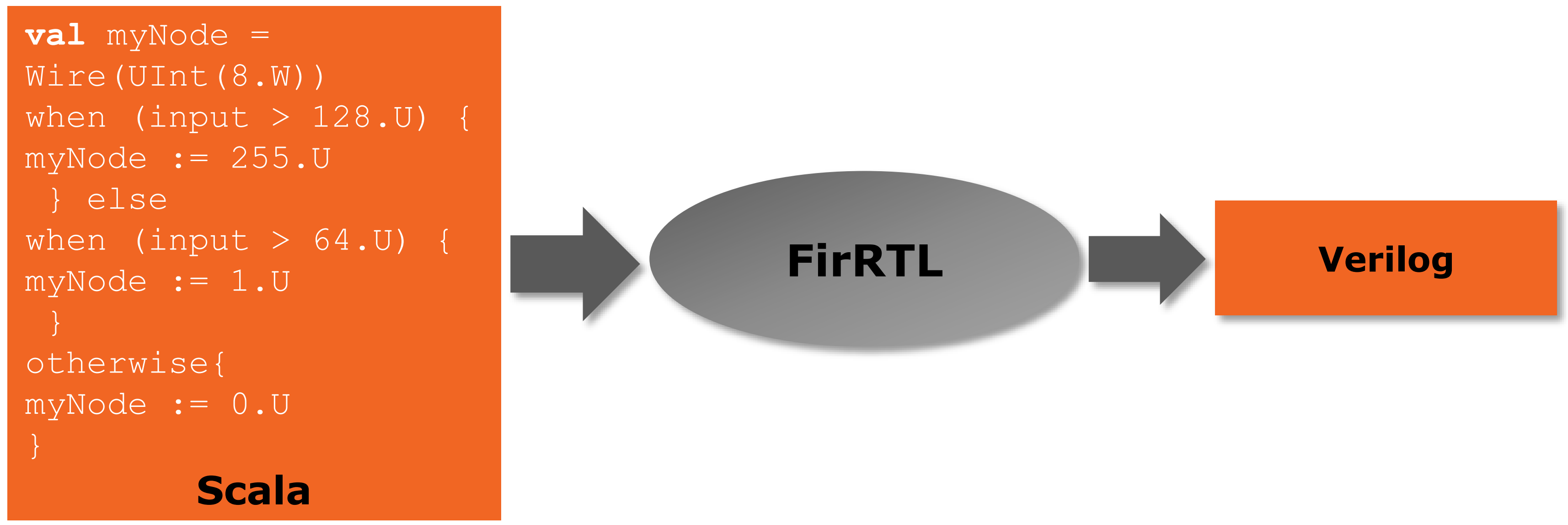
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Solutions

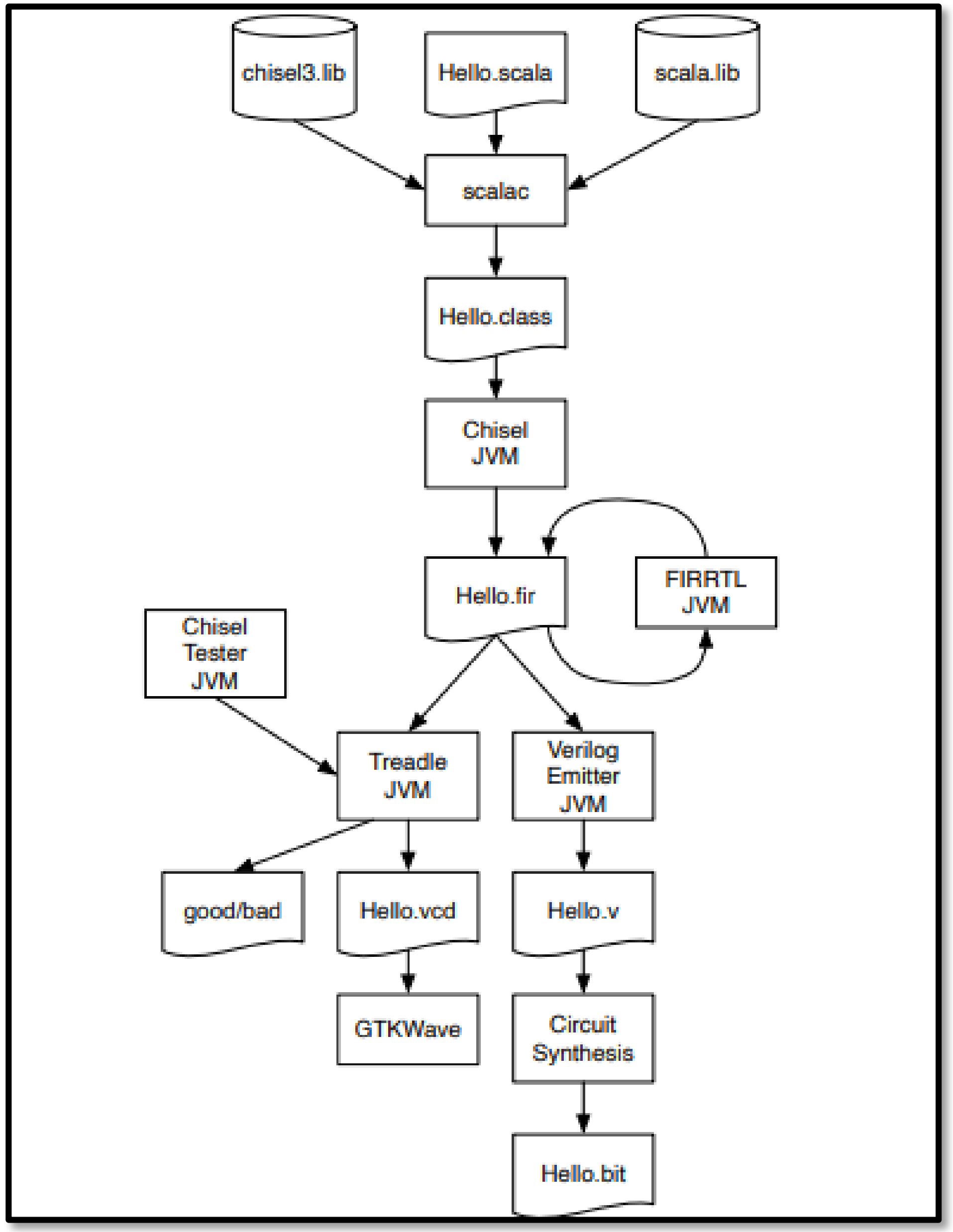
Chisel Based

- Chisel (Constructing Hardware In a Scala Embedded Language) is a hardware design language that facilitates advanced circuit generation and design reuse for both ASIC and FPGA digital logic designs
- Chisel adds hardware construction primitives to the Scala programming language
- Chisel not only allows you to express hardware at the register-transfer level (RTL) but also to write hardware generators
- This generator methodology enables the creation of reusable components and libraries raising the level of abstraction in design while retaining fine-grained control



```
val myNode =
Wire(UInt(8.W))
when (input > 128.U) {
myNode := 255.U
 } else
when (input > 64.U) {
myNode := 1.U
 }
otherwise{
myNode := 0.U
}
```

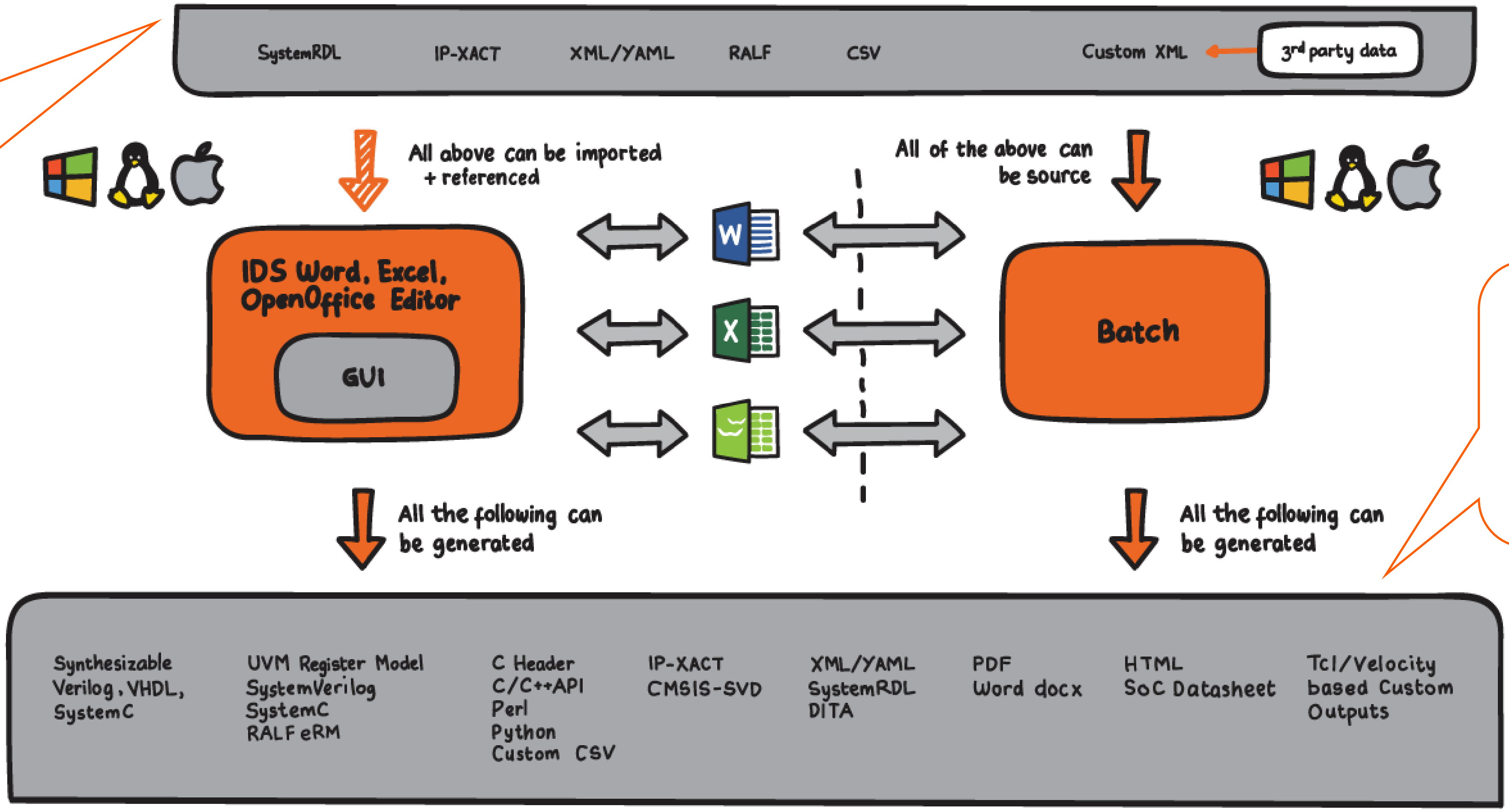
Chisel Based Flow



IDesignSpec™

- **IDesignSpec (IDS)** is a generator that takes a high-level specification and generates parameterizable code
- So, it helps IP/SoC design architects and engineers to create an executable specification for registers and automatically generate output for SW and HW teams

Specifications can be written in **MS Word, MS Excel, LibreOffice** or text-based industry standard formats such as **SystemRDL, RALF or IP-XACT**


SystemRDL
IP-XACT
XML/YAML
RALF
CSV
Custom XML
3rd party data
All above can be imported + referenced
All of the above can be source
IDS Word, Excel, OpenOffice Editor
GUI
Batch
All the following can be generated
All the following can be generated
Synthesizable Verilog, VHDL, SystemC
UVM Register Model SystemVerilog SystemC RALF eRM
C Header C/C++API Perl Python Custom CSV
IP-XACT CMSIS-SVD
XML/YAML SystemRDL DITA
PDF Word docx
HTML SoC Datasheet
Tcl/Velocity based Custom Outputs

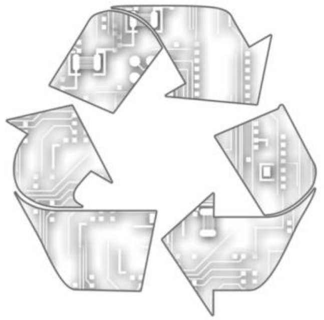
IDesignSpec captures simple as well as **special registers, signals, interrupts, and generates synthesizable RTL, UVM model, C/C++ headers, HTML or PDF**

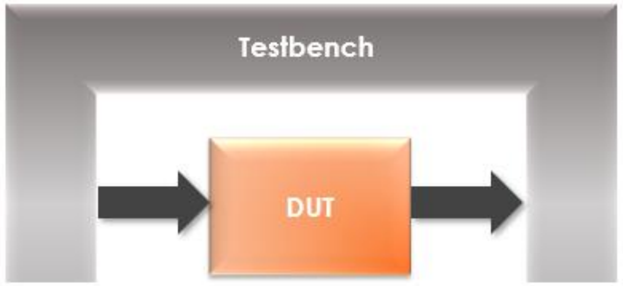
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

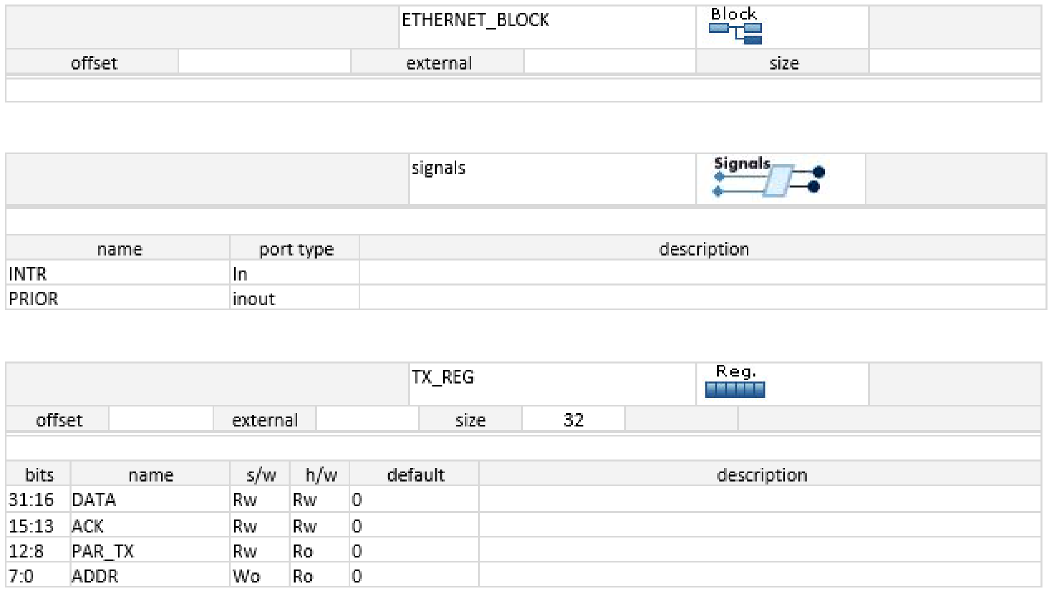
IDesignSpec™ - Continued

Benefits and Capabilities

- Template based approach
- Design reuse
- Eliminate inefficiencies
- Generate hardware and testbench
- A better approach





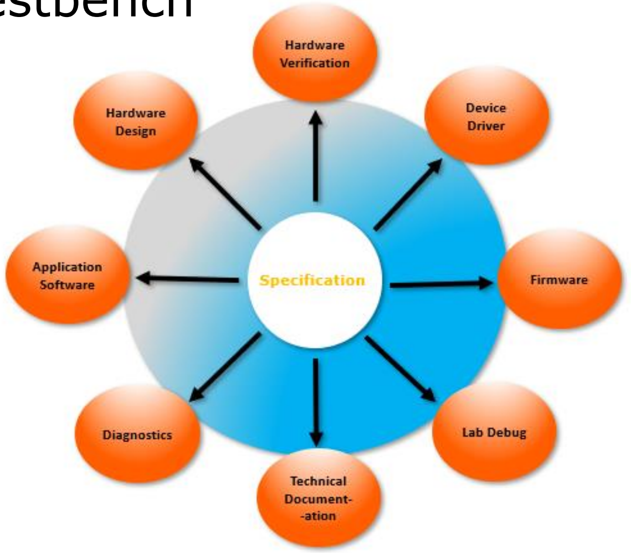


		ETHERNET_BLOCK		Block	
offset		external		size	

	signals	Signals
name	port type	description
INTR	In	
PRIOR	inout	

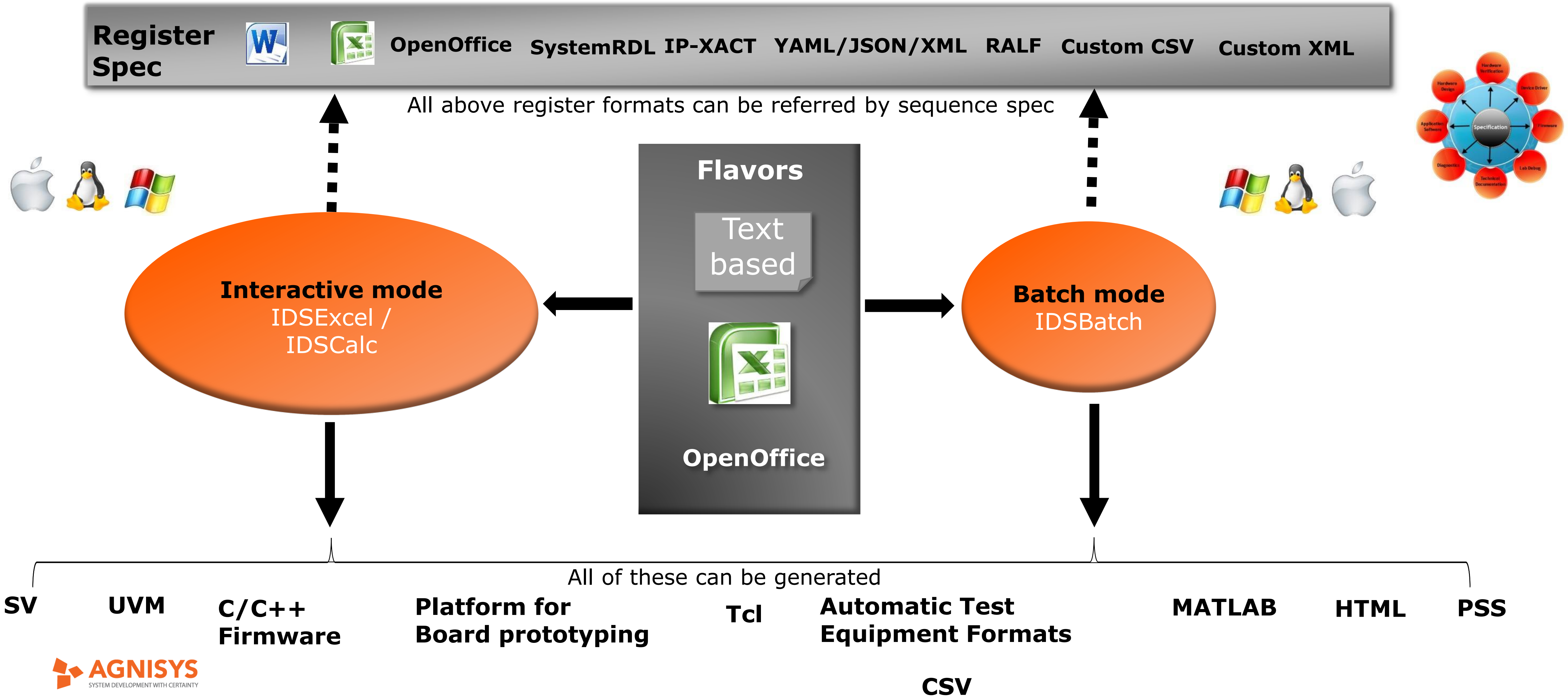
		TX_REG		Reg.	
offset		external		size	32

bits	name	s/w	h/w	default	description
31:16	DATA	Rw	Rw	0	
15:13	ACK	Rw	Rw	0	
12:8	PAR_TX	Rw	Ro	0	
7:0	ADDR	Wo	Ro	0	



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

ISequenceSpec™



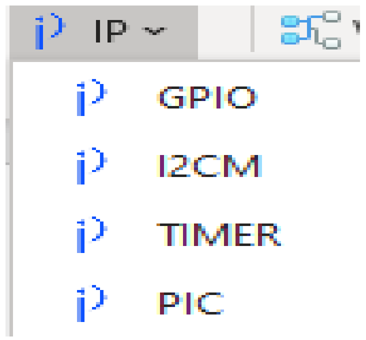
SLIP-G: Standard Library of IP Generators

- Automatically generate standard IPs, while providing add-in functionality of configurability and customizability

- Standard sequences can also be created for the same.

- Standard Agnisys Library:

➢ GPIO
➢ I2C
➢ TIMER
➢ PIC
➢ DMA



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

SLIP-G - Continued

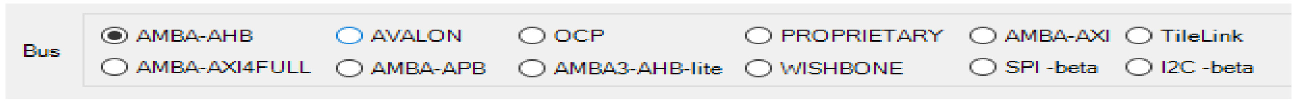
Standard Sequences

- In addition to the register specification for IPs, a set of standard programming sequences can be created for the IPs
- These configuration sequences will help configure the IPs through a few arguments
- These sequences are a standard set of sequences used to initialize and configure the registers of the selected IP
- Sequences will be created for various scenarios/configurations of that IP
- Example
  - In IDS, TIMER IP can be configured in multiple ways by selecting the generation parameters such as the number of timers, number of sources, counter width, prescaler width, etc.

SLIP-G - Continued

Features

- Highly customizable and configurable
- Supports all IDS standard bus interfaces
- Hooks to custom glue logic
- Comes with standard sequences
- Tested, verified and validated IPs



Bus

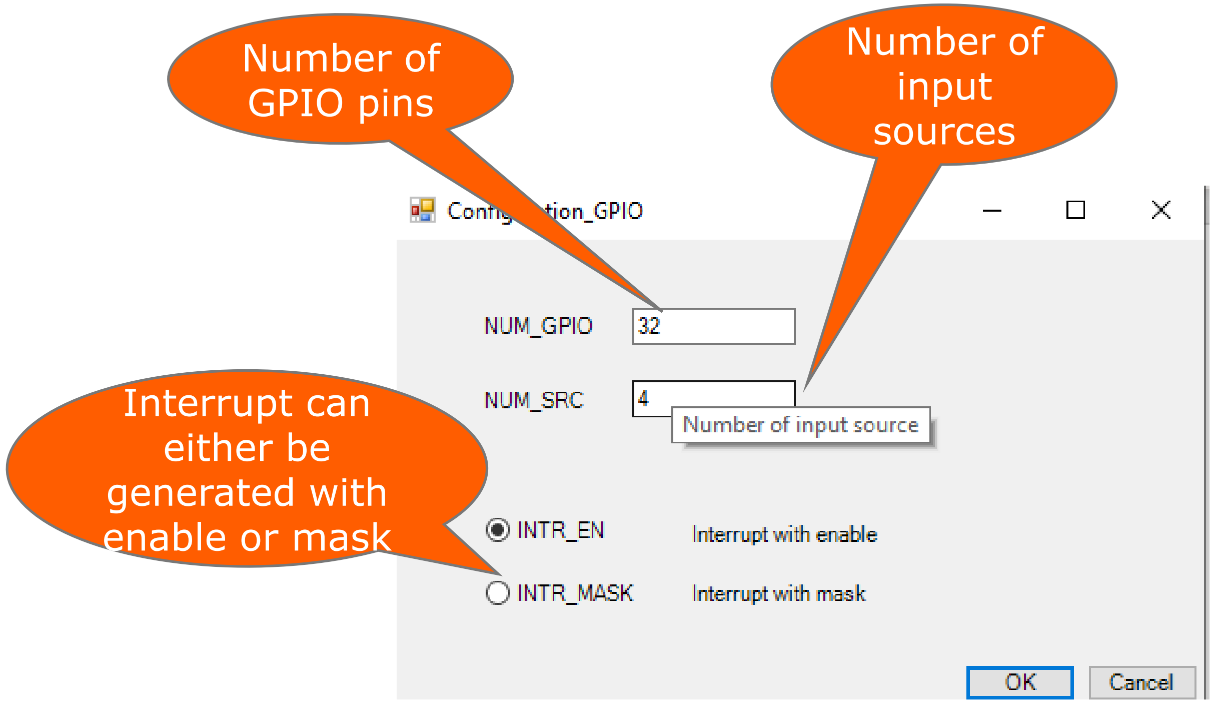
[✓ AMBA-AHB]	AVALON	OCP	PROPRIETARY	AMBA-AXI	TileLink
AMBA-AXI4FULL	AMBA-APB	AMBA3-AHB-lite	WISHBONE	SPI -beta	I2C -beta

SLIP-G Overview

Configurability

GPIO

- GPIO stands for General Purpose Input/Output
- It's a standard interface used to connect microcontrollers to other electronic devices

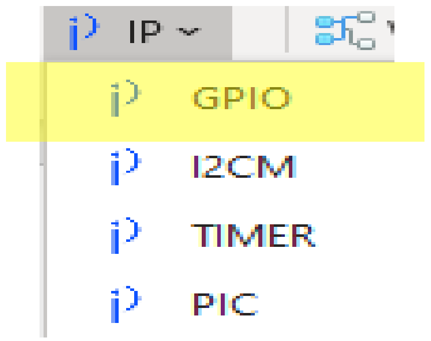


NUM_GPIO 32

NUM_SRC 4 — Number of input source

[✓ INTR_EN] Interrupt with enable

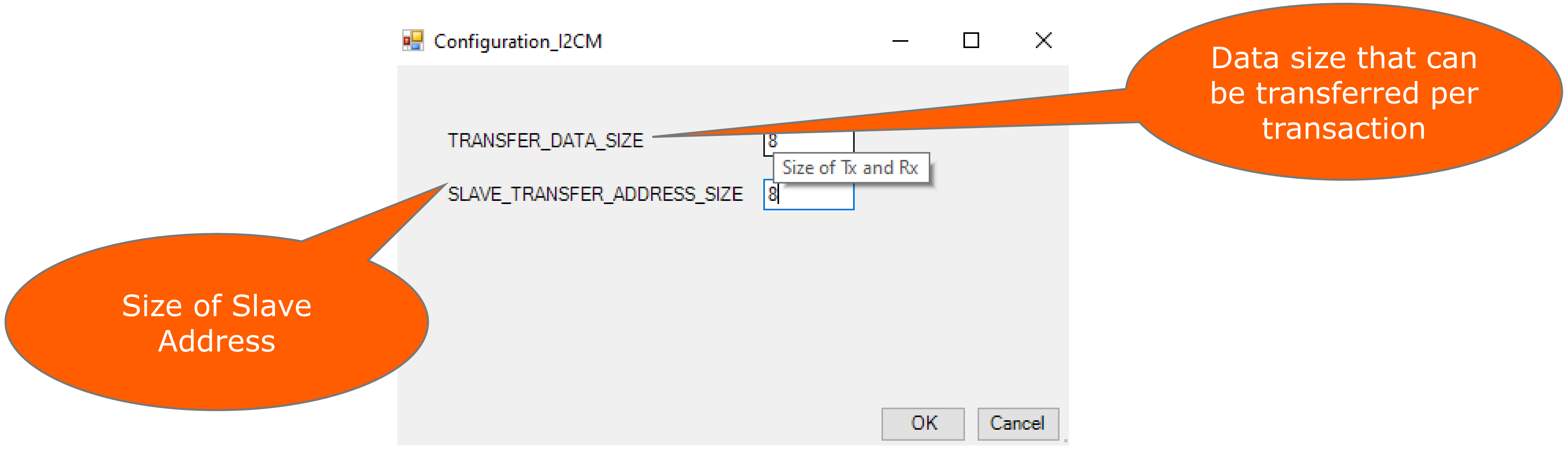
INTR_MASK Interrupt with mask



Configurability – Continued

I2C

- I2C is a serial protocol for a 2-wire interface to connect low-speed devices such as microcontrollers, A/D and D/A converters, EEPROMs, I/O interfaces and other similar peripherals in embedded systems

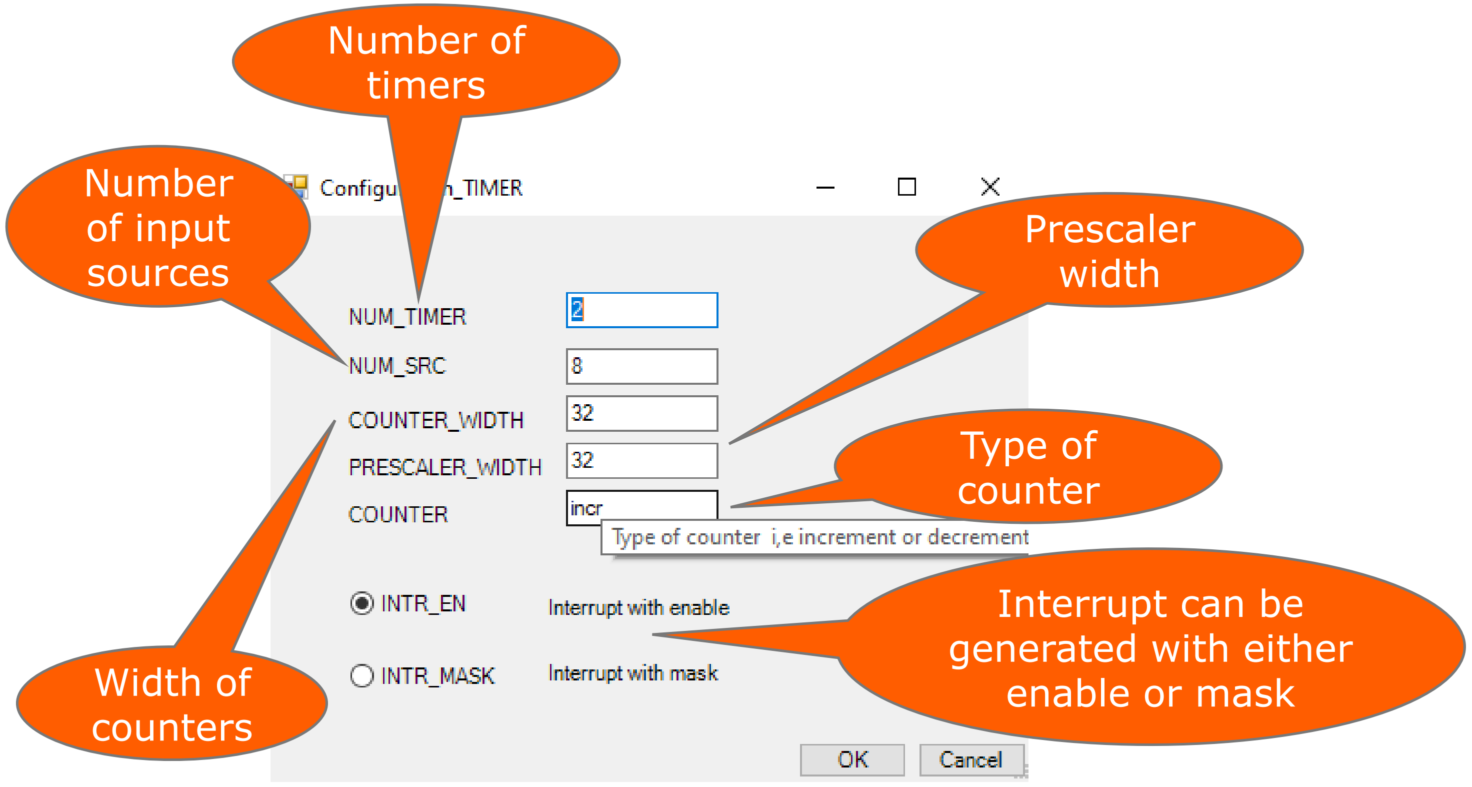


TRANSFER_DATA_SIZE 8
SLAVE_TRANSFER_ADDRESS_SIZE 8

Configurability – Continued

TIMER

- This block is used to provides the timing information
- This block provides the measurement of the pulse width, period of the clock by using external sources



NUM_TIMER	2
NUM_SRC	8
COUNTER_WIDTH	32
PRESCALER_WIDTH	32
COUNTER	incr

Type of counter i,e increment or decrement

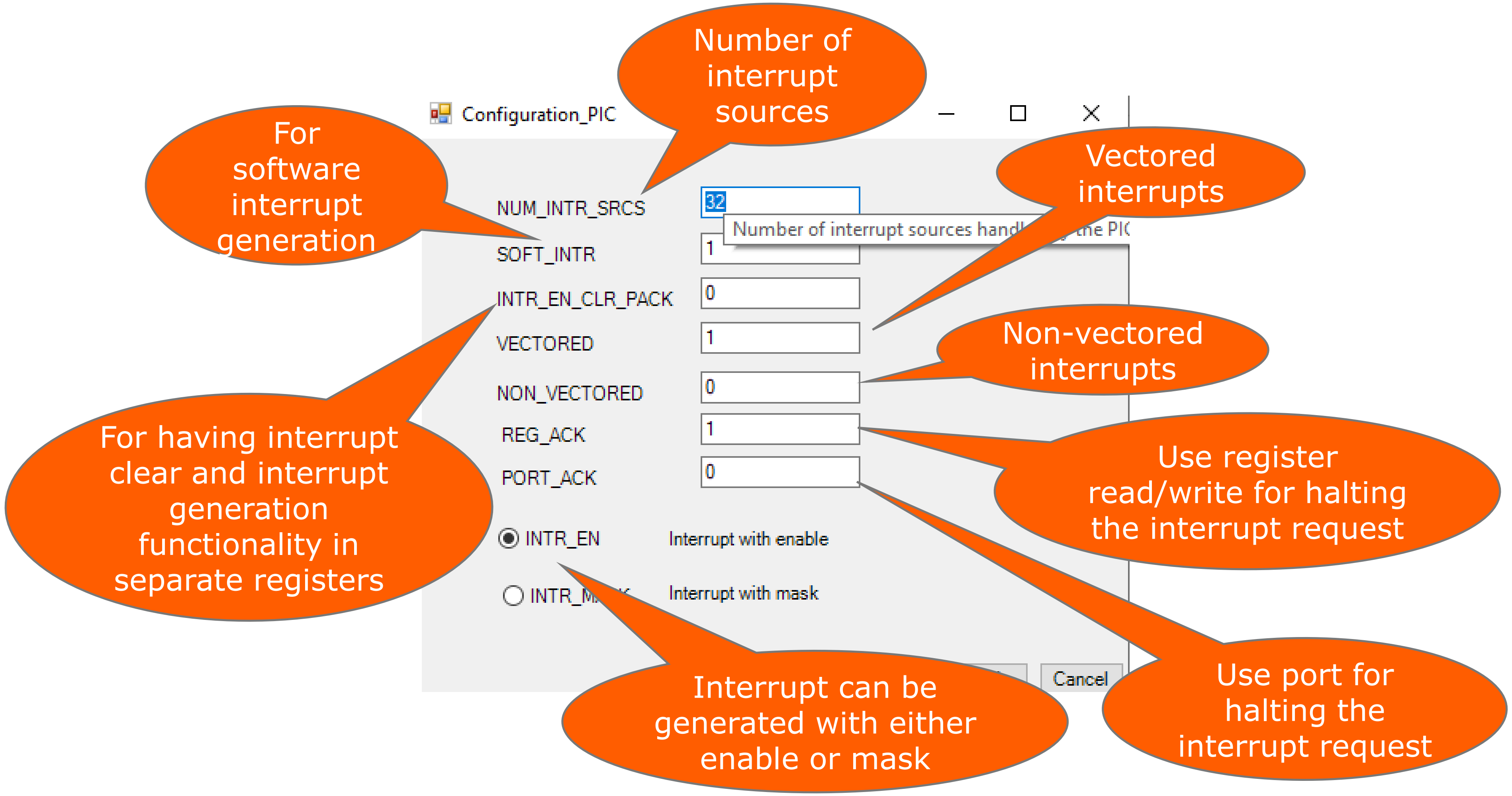
(●) INTR_EN Interrupt with enable

( ) INTR_MASK Interrupt with mask

Configurability – Continued

PIC

- PIC stands for Programmable Interrupt Controller
- This is an external device which takes interrupt sources from various peripherals and depending upon their priorities routes them to one or more CPU lines



Configuration_PIC

NUM_INTR_SRCS	32
SOFT_INTR	1
INTR_EN_CLR_PACK	0
VECTORED	1
NON_VECTORED	0
REG_ACK	1
PORT_ACK	0

[✓ INTR_EN] Interrupt with enable

INTR_M Interrupt with mask

Number of interrupt sources hand the PI

Number of interrupt sources
For software interrupt generation
Vectored interrupts
Non-vectored interrupts
For having interrupt clear and interrupt generation functionality in separate registers
Use register read/write for halting the interrupt request
Interrupt can be generated with either enable or mask
Use port for halting the interrupt request
Cancel

Customizability

- Flexibility to add custom fields to the register
- Flexibility to add custom signals through signal tables
- Flexibility to add custom registers for adding user-specified functionality

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Standard Sequences

- In SLIP-G, user gets the flexibility to create configurable APIs
- Configuration APIs are basically customizable IP initialization sequences to initialize user-controlled registers of the standard IPS through simple arguments
- User can generate configuration APIs for different modes of the IPs using different arguments
- Standard sequences are generated for
  - TIMER
  - GPIO
  - I2C
  - PIC

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Deep dive into SLIP-G

GPIO

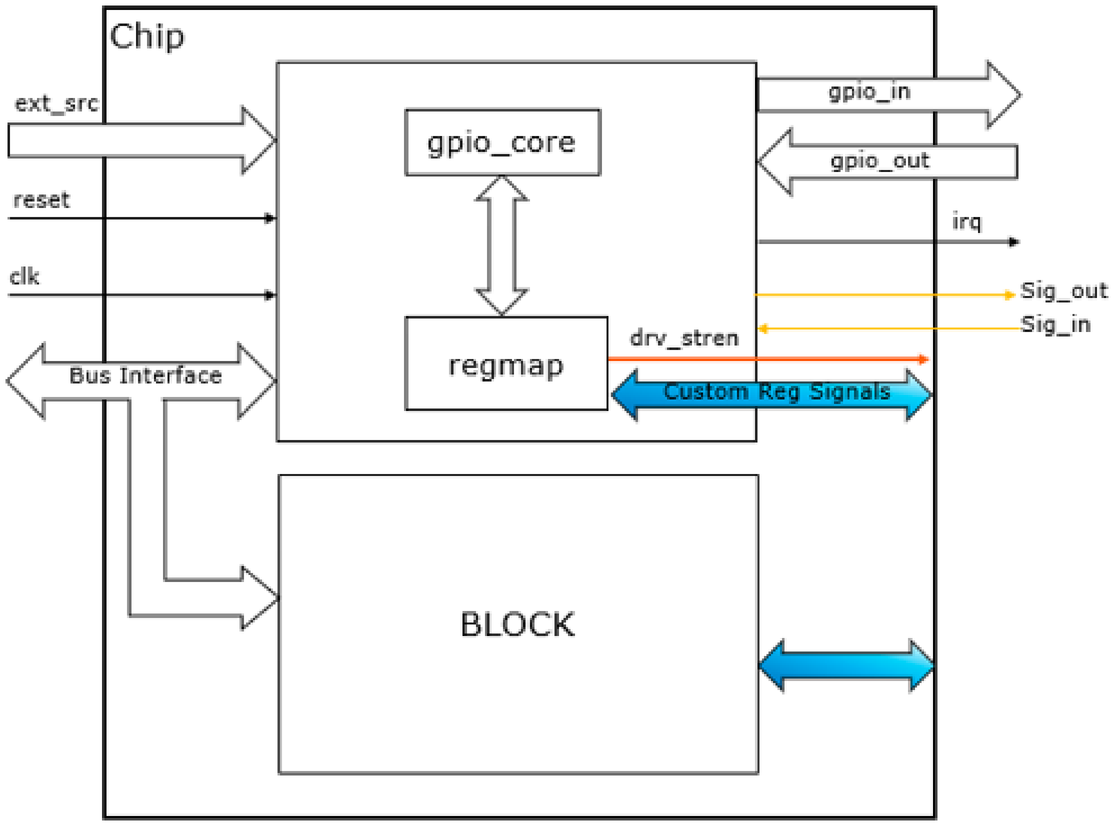
- GPIO stands for General Purpose Input/Output

- It's a standard interface used to connect microcontrollers to other electronic devices
  - For example, it can be used with sensors, diodes, displays, and system-on-chip (SoC) modules

- GPIO pins do not have any predefined functions and they are unused by default
  - These pins' behavior is controlled by the users at run time

- GPIOs can work as either input or output depending upon the configuration setting for each GPIO pin

BLOCK DIAGRAM



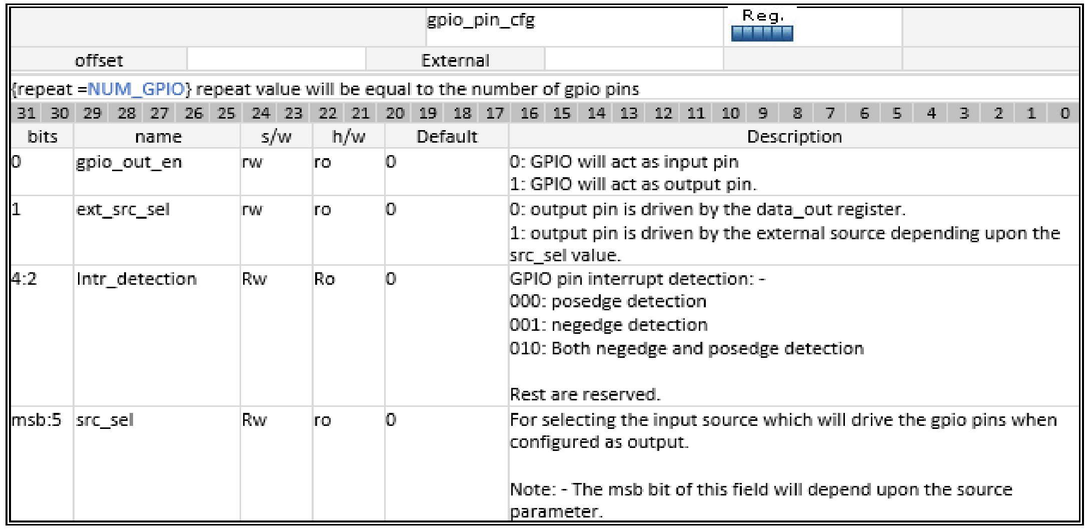
GPIO - Continued

- It has:
  - Configurable number of GPIO pins
  - Configurable number of external sources for driving the GPIO pins
  - Configurable interrupt detection for GPIO pins selected as input at run time
  - For 32 pin GPIO:
    - Maximum operational frequency: 167MHz
    - Size: 1188 LUTs (Xilinx® ZYNQ®)

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

GPIO - Continued

Register Map



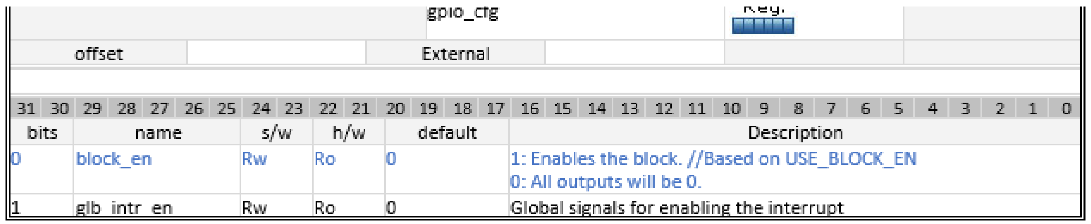
gpio_pin_cfg | Reg.

offset | External

{repeat =NUM_GPIO} repeat value will be equal to the number of gpio pins

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	Default	Description
0	gpio_out_en	rw	ro	0	0: GPIO will act as input pin 1: GPIO will act as output pin.
1	ext_src_sel	rw	ro	0	0: output pin is driven by the data_out register. 1: output pin is driven by the external source depending upon the src_sel value.
4:2	Intr_detection	Rw	Ro	0	GPIO pin interrupt detection: - 000: posedge detection 001: negedge detection 010: Both negedge and posedge detection Rest are reserved.
msb:5	src_sel	Rw	ro	0	For selecting the input source which will drive the gpio pins when configured as output. Note: - The msb bit of this field will depend upon the source parameter.

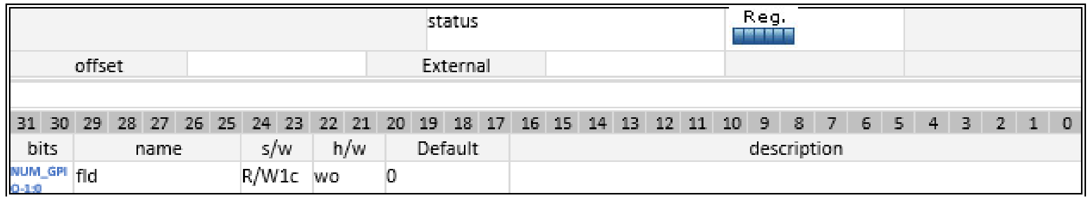


gpio_cfg | Reg.

offset | External

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	Description
0	block_en	Rw	Ro	0	1: Enables the block. //Based on USE_BLOCK_EN 0: All outputs will be 0.
1	glb_intr_en	Rw	Ro	0	Global signals for enabling the interrupt

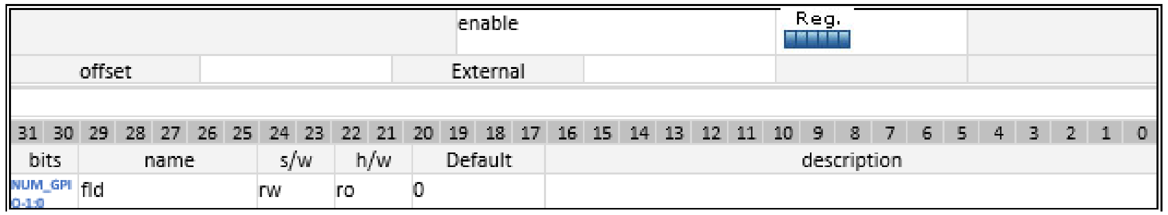


status | Reg.

offset | External

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	Default	description
NUM_GPIO-1:0	fld	R/W1c	wo	0	

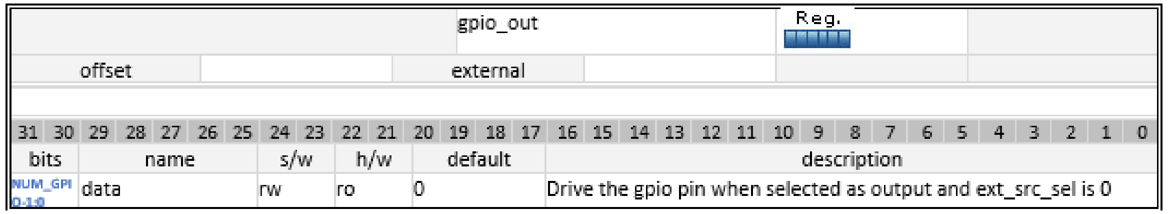


enable | Reg.

offset | External

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	Default	description
NUM_GPIO-1:0	fld	rw	ro	0	

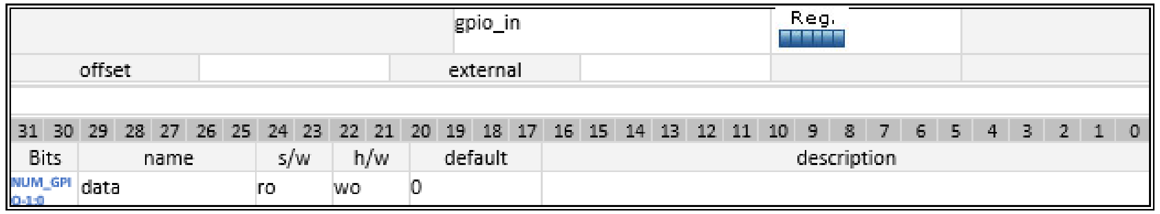


gpio_out | Reg.

offset | external

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
NUM_GPIO-1:0	data	rw	ro	0	Drive the gpio pin when selected as output and ext_src_sel is 0

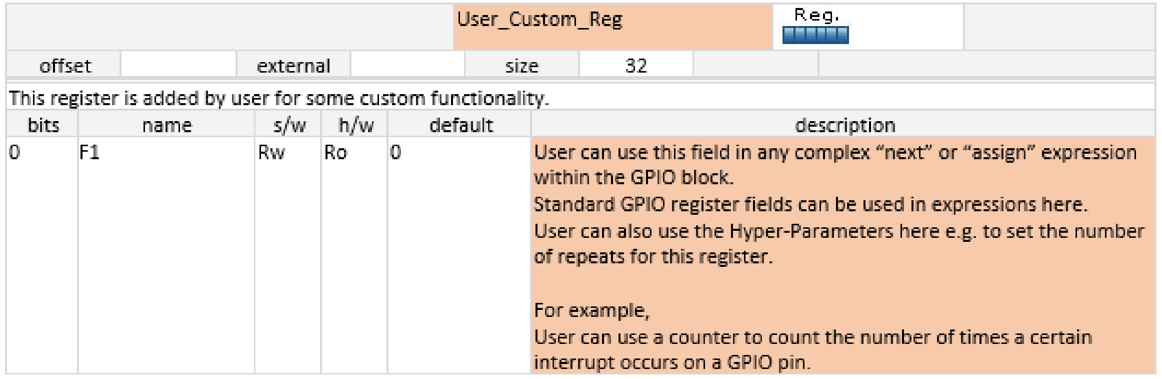


gpio_in | Reg.

offset | external

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	name	s/w	h/w	default	description
NUM_GPIO-1:0	data	ro	wo	0	



User_Custom_Reg | Reg.

offset | external | size 32

This register is added by user for some custom functionality.

bits	name	s/w	h/w	default	description
0	F1	Rw	Ro	0	User can use this field in any complex "next" or "assign" expression within the GPIO block. Standard GPIO register fields can be used in expressions here. User can also use the Hyper-Parameters here e.g. to set the number of repeats for this register. For example, User can use a counter to count the number of times a certain interrupt occurs on a GPIO pin.

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

GPIO - Continued

Standard Sequences

- GPIO pin as output with external source
  - This API is used to configure GPIO pin as an output when driving with external source
  - The following arguments need to be passed while using this API:
    - out_pin - GPIO pin number to be configured as an output
    - out_src_sel - select the external source which will drive the GPIO out pin
    - Usage-

      GPIO_init_out_ext_src(out_pin , ext_src_sel)

- GPIO pin as output with no external source
  - This API is used to configure GPIO pin as an output when driving with gpio_out register
  - The following arguments need to be passed while using this API:
    - out_pin - GPIO pin number to be configured as an output
    - gpio_out_val - drive the gpio pin with gpio_out register
    - Usage-

      GPIO_init_out_no_ext_src(out_pin , gpio_out_val)

GPIO - Continued

Standard Sequences

- GPIO pin as input with enable interrupt with posedge detection
  - This API is used to configure GPIO pin as an input with enable interrupt when interrupt occurs on posedge
  - The following arguments need to be passed while using this API:
    - inp_pin - GPIO pin number to be configured as an input
    - intr_enb - GPIO pin enable interrupt
    - Usage-

      GPIO_init_in_enb_posedge_detect(inp_pin , intr_enb)
    - Similarly, for negedge use **GPIO_init_in_enb_negedge_detect** API

- GPIO pin as input with enable interrupt with both posedge and negedge detection
  - This API is used to configure GPIO pin as an input with enable interrupt when interrupt occurs on both posedge and negedge
  - The following arguments need to be passed while using this API:
    - inp_pin - GPIO pin number to be configured as an input
    - intr_enb - GPIO pin enable interrupt
    - Usage-

      GPIO_init_in_enb_pos_neg_edge(inp_pin , intr_enb)

I2C

- I2C is a serial protocol for a 2-wire interface to connect low-speed devices such as microcontrollers, A/D and D/A converters, EEPROMs, I/O interfaces and other similar peripherals in embedded systems

- The I2C Master is used to communicate between the processor and different slaves in the chip

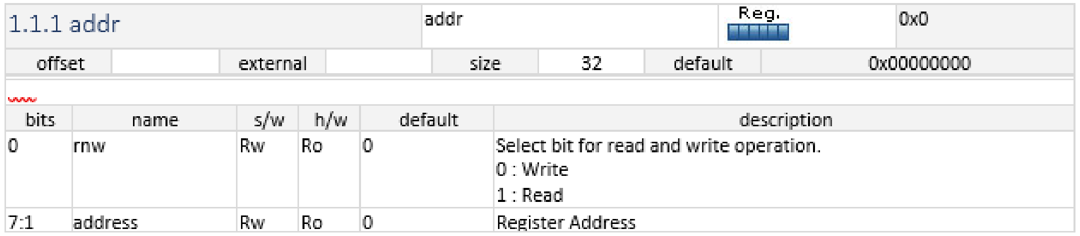
- The I2C in SLIP-G is a subset of the I2C protocol as some of the features of the protocol are not supported

- It has:
  - Configurable sizes of read and write data transactions
  - Configurable operational clock frequency
  - Interrupts for various events
  - I2C with transfer data size of 16 uses 296 LUTs (Xilinx® ZYNQ®)

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

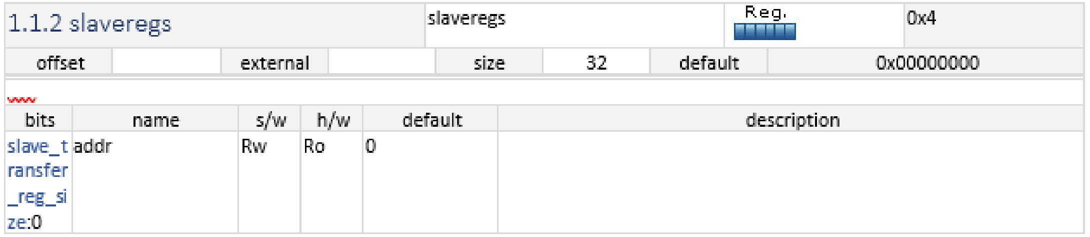
I2C - Continued

Register Map



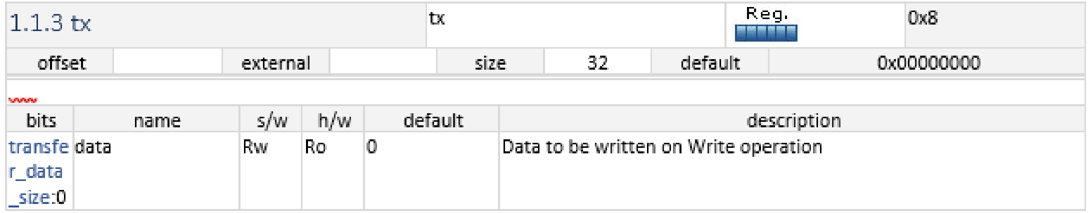
1.1.1 addr			addr		Reg.	0x0	
offset		external		size	32	default	0x00000000

bits	name	s/w	h/w	default	description
0	rnw	Rw	Ro	0	Select bit for read and write operation. 0 : Write 1 : Read
7:1	address	Rw	Ro	0	Register Address



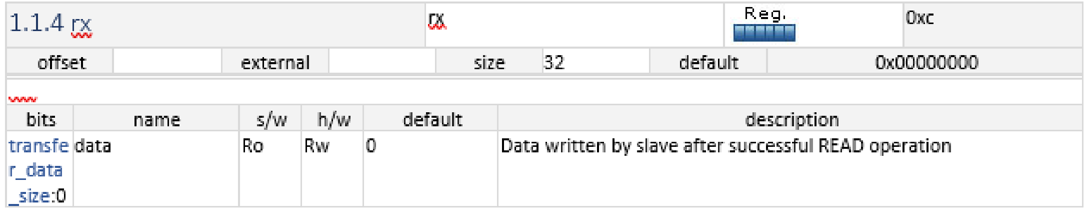
1.1.2 slaveregs			slaveregs		Reg.	0x4	
offset		external		size	32	default	0x00000000

bits	name	s/w	h/w	default	description
slave_transfer_reg_size:0	addr	Rw	Ro	0	



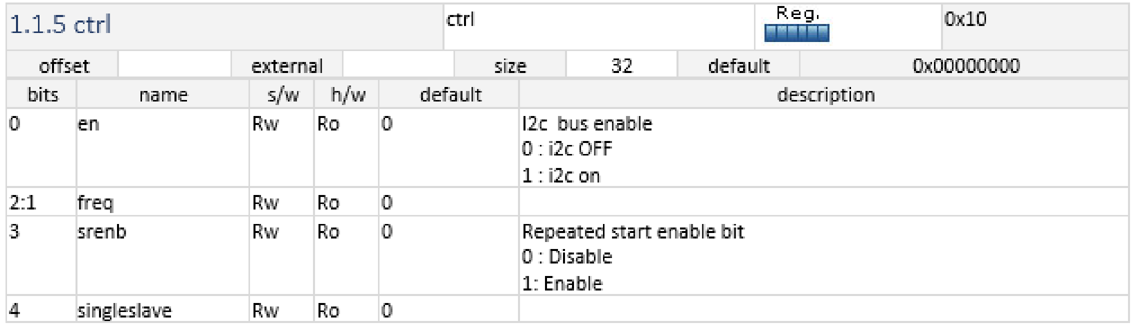
1.1.3 tx			tx		Reg.	0x8	
offset		external		size	32	default	0x00000000

bits	name	s/w	h/w	default	description
transfer_data_size:0	data	Rw	Ro	0	Data to be written on Write operation



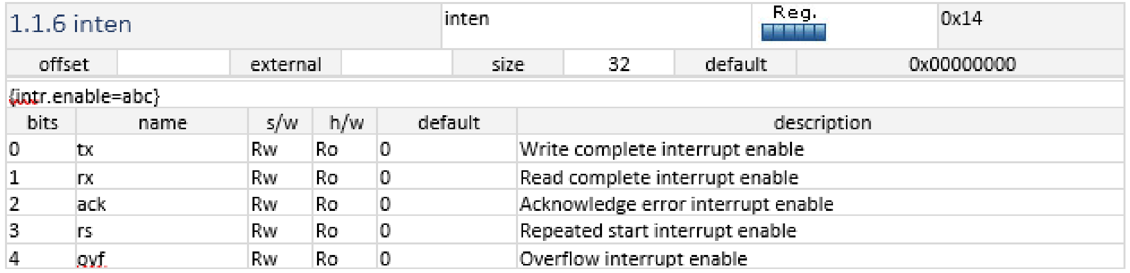
1.1.4 rx			rx		Reg.	0xc	
offset		external		size	32	default	0x00000000

bits	name	s/w	h/w	default	description
transfer_data_size:0	data	Ro	Rw	0	Data written by slave after successful READ operation



1.1.5 ctrl			ctrl		Reg.	0x10	
offset		external		size	32	default	0x00000000

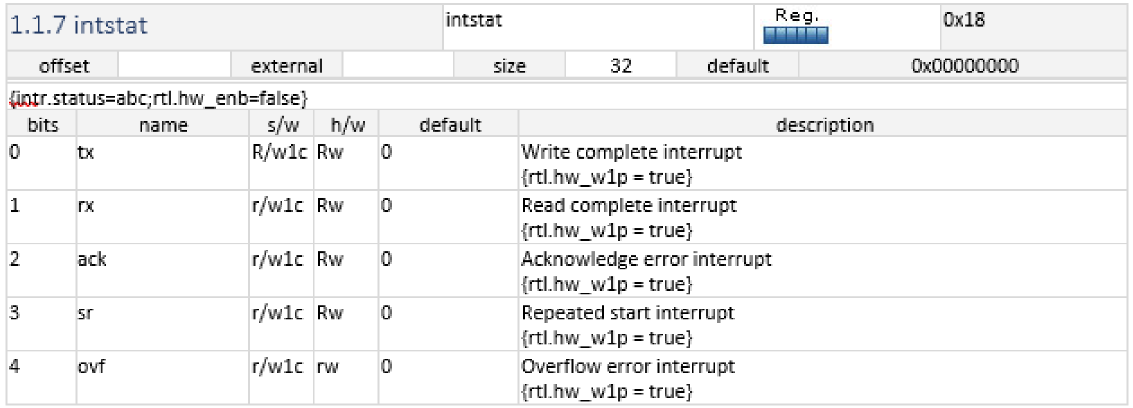
bits	name	s/w	h/w	default	description
0	en	Rw	Ro	0	I2c bus enable 0 : i2c OFF 1 : i2c on
2:1	freq	Rw	Ro	0	
3	srenb	Rw	Ro	0	Repeated start enable bit 0 : Disable 1: Enable
4	singleslave	Rw	Ro	0	



1.1.6 inten			inten		Reg.	0x14	
offset		external		size	32	default	0x00000000

{intr.enable=abc}

bits	name	s/w	h/w	default	description
0	tx	Rw	Ro	0	Write complete interrupt enable
1	rx	Rw	Ro	0	Read complete interrupt enable
2	ack	Rw	Ro	0	Acknowledge error interrupt enable
3	rs	Rw	Ro	0	Repeated start interrupt enable
4	ovf	Rw	Ro	0	Overflow interrupt enable



1.1.7 intstat			intstat		Reg.	0x18	
offset		external		size	32	default	0x00000000

{intr.status=abc;rtl.hw_enb=false}

bits	name	s/w	h/w	default	description
0	tx	R/w1c	Rw	0	Write complete interrupt {rtl.hw_w1p = true}
1	rx	r/w1c	Rw	0	Read complete interrupt {rtl.hw_w1p = true}
2	ack	r/w1c	Rw	0	Acknowledge error interrupt {rtl.hw_w1p = true}
3	sr	r/w1c	Rw	0	Repeated start interrupt {rtl.hw_w1p = true}
4	ovf	r/w1c	rw	0	Overflow error interrupt {rtl.hw_w1p = true}

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

I2C - Continued

Standard Sequences

- i2cReset
  - This API clears the registers of the 'i2cm' IP specification, including the addr, slaveregs, tx, ctrl, inten, intstat, etc.

- i2cWrite
  - This API is used to configure the i2cm for a write transaction in the slave
  - The arguments to passed for calling this API are:
    - slave_addr - to pass the base address of the slave to perform the write transaction
    - reg_addr - to pass the register address of the slave to perform the write transaction
    - data_write - to pass the data that is to be written in the transaction
    - wr_intr_enb - to enable write interrupt
    - Usage-

```
i2cwrite( slave_addr , reg_addr , data_write , wr_intr_enb )
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

I2C - Continued

Standard Sequences

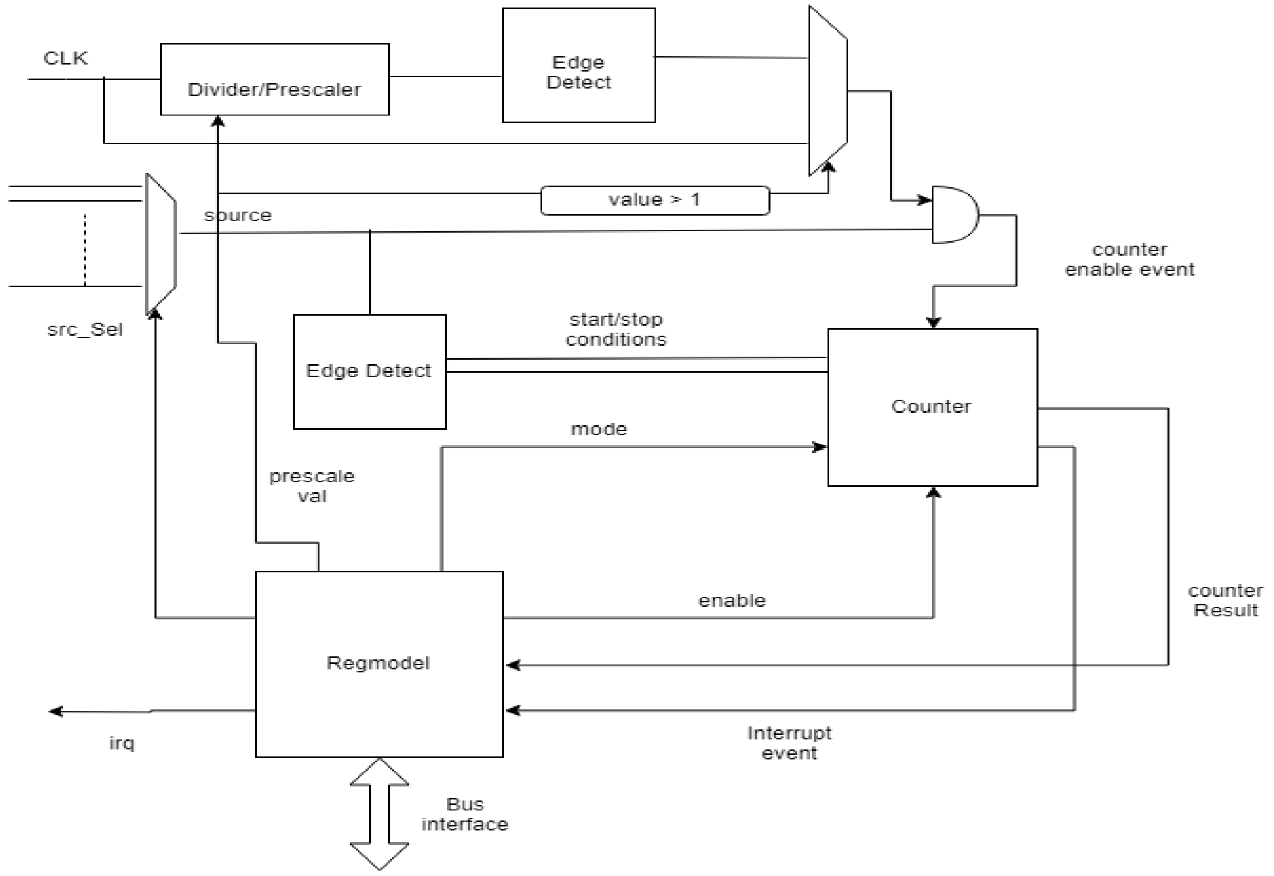
- i2cRead
  - This API is to configure the i2cm for a read transaction in the slave
  - The arguments to passed for calling this API are :
    - slave_addr - to pass the base address of the slave to perform the read transaction
    - reg_addr - to pass the register address of the slave to perform the read transaction
    - Usage-

      i2cread (slave_addr, reg_addr)

TIMER

- This block is used to provides the timing information or timing events
- It is sometimes also used to measure pulse width, period of the clock by using external sources
- The timer is an increment counter whose width and counting events will be selected through generation parameters
- Functionalities offered by the block are:
  - Free running mode
  - Periodic mode
  - Prescaling
  - Interrupt generation

BLOCK DIAGRAM



AGNISYS SYSTEM DEVELOPMENT WITH CERTAINTY

TIMER - Continued

- It has:
  - Configurable number of timers
  - Selectable counter types - increment or decrement
  - Configurable counter width with support for wide counters
  - Configurable measurement modes at run time
  - Interrupt generations for different modes
  - Configurable pre-scaler at run time
  - For 32-bit increment counter with 8 number of sources:
    - Maximum operational frequency: 150MHz
    - Size: 1149 LUTs (Xilinx® ZYNQ®)

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

TIMER - Continued

Register Map

	Control	Reg.	
offset		external	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	name	s/w	h/w	default	description
0	en	rw	ro	0	1:- Enables the timer block or calculations 0:- Disables the timer block
2:1	mode	rw	ro	0	00: running mode 01: Periodic mode with source enable 10: reserved 11: reserved
5:3	Event_sel	rw	ro	0	Legal values for running mode 000: high level 001: low level 010: between two high edges 011: between two low edges Legal values for the Periodic Mode 100: Posedges 101: negedges 110: both edges 111: without the source and counting event will be the posedge of the clock
Src_width-1:0	Src_sel	rw	ro	0	To select the source on which measurement will be performed. These act as a counter enable signals for a given timer. If no source width is defined, then the timer will count the clock edges depending upon the prescaling value.

	period	Reg.	
offset		external	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	name	s/w	h/w	default	description
31:0	F1	rw	ro	0	

	Result	Reg.	
offset		external	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits	name	s/w	h/w	default	description
31:0	data	ro	wo	0	

	Status	Reg.	
offset		external	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
0	Overflow/underflow	r/w1c	wo	0	
1	run_intr	r/w1c	wo	0	
2	period_intr	r/w1c	wo	0	
3	cfg_ch	r/w1c	wo	0	

	Enable	Reg.	
offset		external	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
0	Over_intr_en	rw	na	0	
1	Run_intr_en	rw	na	0	
2	Period_intr_en	rw	na	0	

	Prescaler	Reg.	
offset		external	

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

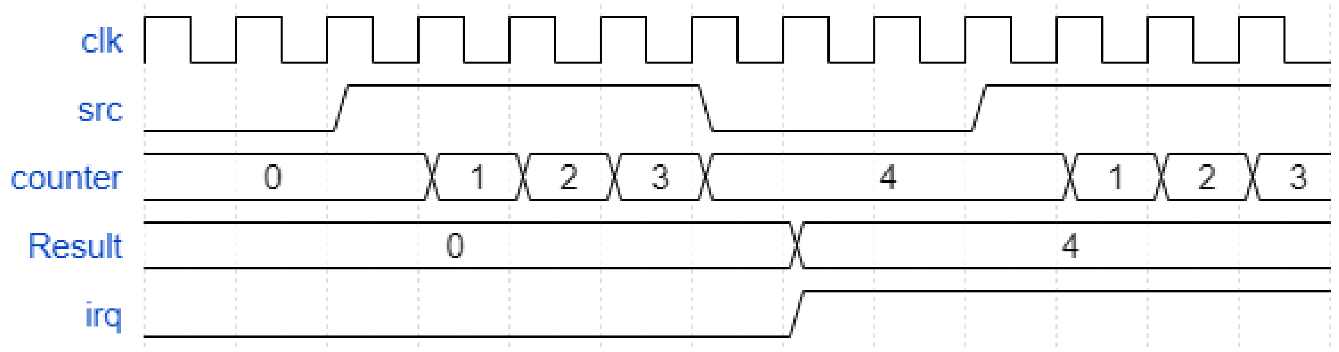
bits	name	s/w	h/w	default	description
Prescale_width:0	F1	rw	ro	0	Defines the prescaling values.

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

TIMER - Continued

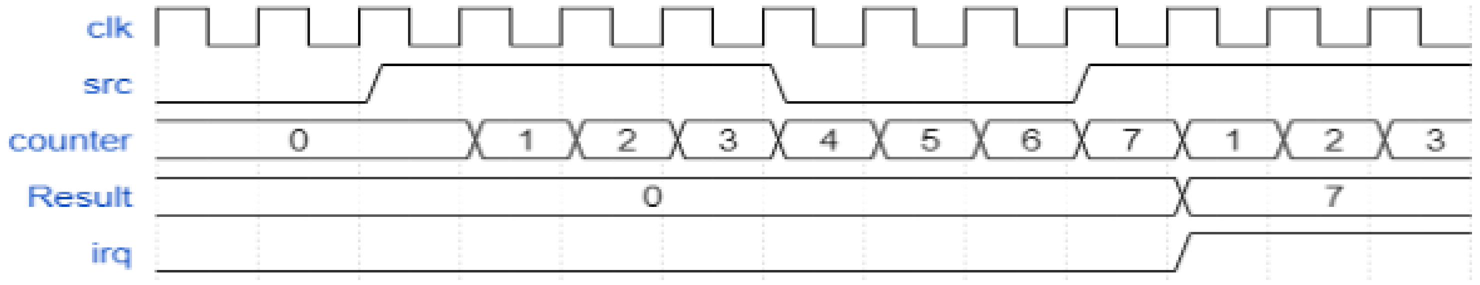
Modes of operation

- Running mode with high level



Waveform for Increment Counter for high level signal

- Running mode with two consecutive high level

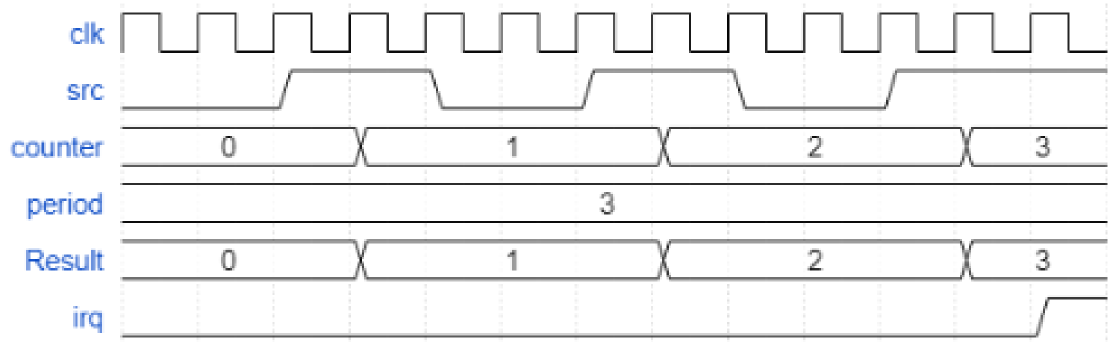


Waveform for Increment Counter between two posedges

TIMER - Continued

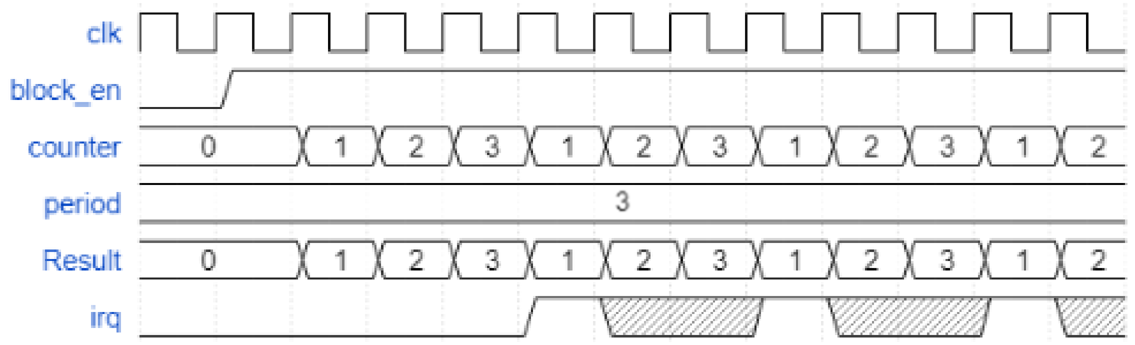
Modes of operation

- Periodic mode with source



Waveform for Increment Counter when the counting event is +ve edge of the selected source

- Periodic mode without source



Waveform for increment counter when the selected source is continuously high

TIMER - Continued

Standard Sequences

- reset_seq

  This API clears the registers of the 'Timer' IP specification, including the control register, period register, prescaler register, and counter register

- timer_init_gen_seq
  - This API is a general configuration API to initialize the timer IP with few arguments
  - The arguments to be passed while calling function 'timer_init_gen_seq' are:
    - timer_en – this is a single bit argument to enable the timer
    - timer_mode – this is a two-bit argument to select the mode of the timer
    - timer_event_sel - to select legal event of the selected mode of the timer
    - timer_src_sel - to select the source in which timing measurement is to be performed
    - timer_prescal - to select the 'Prescaler' register value of the Timer IP which modifies the clock frequency for counting events
    - Usage-

      timer_init_gen_seq (timer_en,timer_mode,timer_event_sel,timer_src_sel,timer_prescal)

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

TIMER - Continued

Standard Sequences

- running_mode_high_event
  - This API is to initialize a timer with running mode and 'high level' as 'event select'
  - The arguments to be passed while calling function 'running_mode _high_event' are:
    - timer_src_sel - to select the source to perform 'high level' event of the running mode of the timer.
    - timer_prescal - to select the 'Prescaler' register value of the Timer IP.
    - Usage-

      running_mode_high_event(timer_src_sel , timer_prescal)

- Similarly, there are APIs for running_mode_low_event, running_mode_two_high_event, running_mode_two_low_event, periodic_mode_posedges_event, periodic_mode_negedges_event, periodic_mode_bothedges_event, and default_event

PIC

- PIC stands for Programmable Interrupt Controller

- This is an external device that takes interrupt sources from various peripherals and, depending upon their priorities, routes them to one or more CPU lines

- PIC commonly consists of hard priorities, configurable software priorities

- By using these relative priorities, interrupt source with highest priority is routed to CPU line

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

PIC - Continued

- It has:

  - Configurable number of interrupt sources and their priorities
  - Configurable software interrupt generation
  - Configurable output interrupt generation, including active high or active low and edge or level triggered at run time
  - Configurable enable/disable functionality - packed enable bits in a register or separate enable register for each interrupt source
  - Configurable interrupt clear functionality - interrupt clear bits are packed or present in separate registers
  - Vectored addressing
  - Non vectored IP with 32 interrupt sources and registered acknowledge:
    - Maximum operational frequency: 167MHz
    - Size: 1253 LUTs (Xilinx® ZYNQ®)

PIC - Continued

Register Map

intr_cfg — Reg.

offset | external

{repeat = NUM_INTR_SRCS }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	Default	description
0	detect	rw	Ro	0	Edge detection or level detection
1	level	rw	ro	0	Active high or active low-level interrupt sources

#else

Status — Reg.

offset | external

{repeat = NUM_INTR_SRCS }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	Name	s/w	h/w	Default	description
0	Fld	r/w1c	wo	0	{intr.status=a}

#ifdef UNPACK ==0

Enable — Reg.

offset | external

{repeat = (NUM_INTR_SRCS %32==0) ? NUM_INTR_SRCS /32 : NUM_INTR_SRCS /32 +1 }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	Default	description
31:0	fld	rw	na	0	{intr.enable=a}

#ifdef UNPACK ==0

pending — Reg.

offset | external

{repeat = (NUM_INTR_SRCS %32==0) ? NUM_INTR_SRCS /32 : NUM_INTR_SRCS /32 +1 }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	Default	description
31:0	fld	ro	ro	0	{intr.pending=a}

post — Reg.

offset | external

{repeat = (NUM_INTR_SRCS %32==0) ? NUM_INTR_SRCS /32 : NUM_INTR_SRCS /32 +1 }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
31:0	fld	rw	na	0	{intr.post=a}

priority_reg — Reg.

offset | external

{repeat = NUM_INTR_SRCS }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
Ceil(log2(NUM_INTR_SRCS))-1:0	fld	rw	ro	0	Priorities associated with each interrupt source

intr_addr — Reg.

offset | external

{repeat= NUM_INTR_SRCS }

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
31:0	fld	rw	wo	0	Base address of the vectored table

ISR_addr — Reg.

offset | external

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

bits	name	s/w	h/w	default	description
31:0	fld	ro	na	0	Effective interrupt service routine address of an interrupt source

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

PIC - Continued

Standard Sequences

- pic_vectored_negedge: This API handles vectored interrupts when negedge occurs on the interrupt source
- pic_vectored_posedge: This API handles vectored interrupts when posedge occurs on the interrupt source
- pic_vectored_level_low: This API handles vectored interrupts when the interrupt source has active low signal
- pic_vectored_level_high: This API handles vectored interrupts when the interrupt source has active high

PIC - Continued

Standard Sequences

- pic_non_vectored_level_low : This API handles non vectored interrupts when the interrupt source has active low signal
- pic_non_vectored_level_high: This API handles non vectored interrupts when the interrupt source has active high signal
- pic_non_vectored_posedge: This API handles non vectored interrupts when posedge occurs on the interrupt source
- pic_non_vectored_negedge: This API handles non vectored interrupts when negedge occurs on the interrupt source

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

DMA

- Configurable number of channels
- Interrupt controller for status and diagnostics
- Configurable source and destination address descriptors
- Scatter/Gather – supports non-contiguous data block transfers to a contiguous segment of memory and vice versa
- Round robin arbitration between channels
- Support for multiple modes: memory to memory, memory to peripheral, peripheral to memory, and peripheral to peripheral modes

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Benefits

SLIP-G Benefits

- Fully configurable
  - Supports all configuration parameters for a varied set of needs
- Easily customizable
  - Users can perform customizations such as: add fields to existing registers, add additional registers, or even have dependencies on events of the IP and add arbitrary logic to the IP
- On-the-fly generation
  - IPs are generated from the command line or on a click of a button
- Unencrypted code
  - All the generated files are available as plain text for easy debugging and use by downstream tools
- IPs can be instantiated in IDesignSpec and come with standard sequences for ISS

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

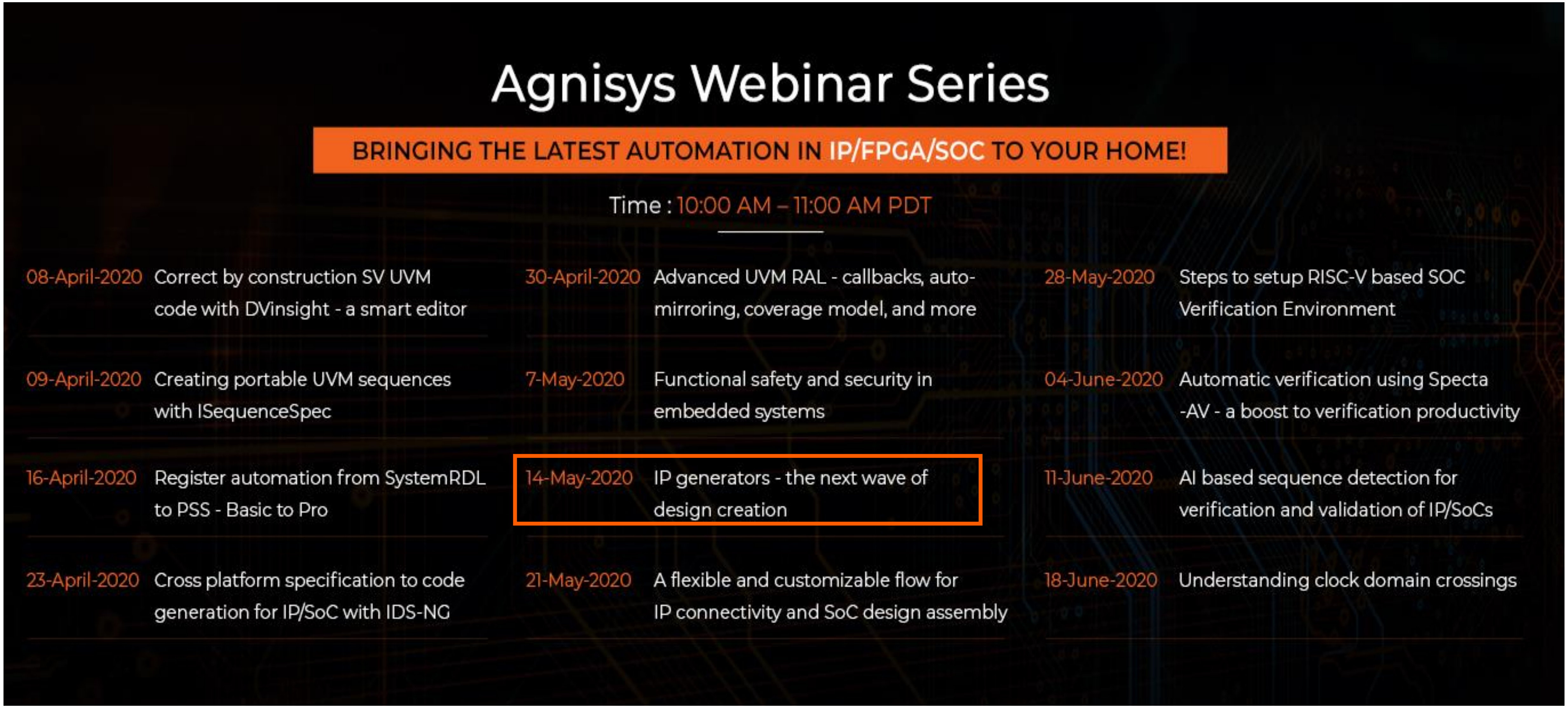
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Conclusion

Conclusion

- Complexity can be handled by using abstraction
- One of the forms of abstraction is reuse
- Reuse is possible if the IPs are customizable and configurable
- Its not just about RTL reuse - reuse must also happen in firmware and software
- Invest in tools and flows that help in the design abstraction and reuse
- Agnisys can help: IDesignSpec^{TM}, ISequenceSpec^{TM}, SLIP-G

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY



Agnisys Webinar Series

BRINGING THE LATEST AUTOMATION IN IP/FPGA/SOC TO YOUR HOME!

Time : 10:00 AM – 11:00 AM PDT

08-April-2020	Correct by construction SV UVM code with DVinsight - a smart editor	30-April-2020	Advanced UVM RAL - callbacks, auto-mirroring, coverage model, and more	28-May-2020	Steps to setup RISC-V based SOC Verification Environment
09-April-2020	Creating portable UVM sequences with ISequenceSpec	7-May-2020	Functional safety and security in embedded systems	04-June-2020	Automatic verification using Specta -AV - a boost to verification productivity
16-April-2020	Register automation from SystemRDL to PSS - Basic to Pro	14-May-2020	IP generators - the next wave of design creation	11-June-2020	AI based sequence detection for verification and validation of IP/SoCs
23-April-2020	Cross platform specification to code generation for IP/SoC with IDS-NG	21-May-2020	A flexible and customizable flow for IP connectivity and SoC design assembly	18-June-2020	Understanding clock domain crossings

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

About Agnisys

- The EDA leader in solving complex design and verification problems associated with HW/SW interface
- Formed in 2007
- Privately held and profitable
- Headquarters in Boston, MA
  - ~1000 users worldwide
  - ~50 customer companies
- Customer retention rate ~90%
- R&D centers (US and India)
- Support centers - committed to ensure comprehensive support
  - Email : support@agnisys.com
  - Phone : 1-855-VERIFYY
  - Response time within one day; within hours in many cases
  - Multiple time zones (Boston MA, San Jose CA and Noida India)

IDESIGNSPEC™ (IDS) EXECUTABLE REGISTER SPECIFICATION

Automatically generate UVM models, Verilog/VHDL models, coverage model, software model, etc.

AUTOMATIC REGISTER VERIFICATION (ARV)

ARV-Sim™ : Create UVM test environment, sequences, and verification plans, and instantly know the status of the verification project

ARV-Formal™ : Create formal properties and assertions, and coverage model from the specification

ISEQUENCESPEC™ (ISS)

Create UVM sequences and firmware routines from the specification

DVinsight™ (DVi)

Smart editor for SystemVerilog and UVM projects

IDS – Next Generation™ (IDS-NG)

Comprehensive SoC/IP spec creation and code generation tool

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY