Reconfigurable Architecture (6)

osana@eee.u-ryukyu.ac.jp

Previously in this class…

- Verilog-HDL syntax and design
  - Circuit description, Module hiearchy, Testbench, parameter…
  - Design flow for simulation and implementation

Productivity in Software Design

- Design reuse
  - Once written, use again and again: write a clean code!
- Share codes if everyone want it
  - Libraries: printf(), STL, BLAS, FFTW, OpenCV, …

Examples of Libraries in Software

- Mathematics: fast, optimized implementations of frequently-used algorithms
  - BLAS (Basic Linear Algebra Subprograms)
  - FFTW (Fastest Fourier Transform in the West)
- Image processing: read/write various image formats, or do transformations
  - OpenJPEG, libPNG, ImageMagick, OpenCV, …

Same in Hardware Design

- Example of complicated stuff: Floating-point arithmetics
  - 1 bit sign x 52bit fraction x $2^{(11\text{bit exponent})}$
  - Fraction is always 1.xxxx (must be adjusted with exponent)
  - Exponent biased by 1023 (exponent=1023 means $x2^0$)
- Fraction and exponent must be calculated separately and together :(
  - Never want to implement FP operators again and again…

Software vs Hardware

- Connecting software components are simple
  - Argument, function call and return value
- In hardware, procedure and timing is crucial
  - Protocol in time-space is important
  - Verification is difficult and time consuming

Design easier with IP cores

- IP (Intellectual Property) cores: functional blocks
  - Blackbox modules (sometimes source code is provided, but usually not)
    - Simulation with IP cores is possible
  - **Interface specification** is important and well documented
  - Many IP core comes with **user-configurable parameters**

Hard IP vs Soft IP

- Soft IP: Constructed on FPGA's logic cells, as our HDL is mapped
  - Flexible in number, location and functionality
  - May be a pressure on logic utilization
- Hard IP: "Ready-made" function block on FPGA chip
  - Memory blocks, DSP arithmetics, Ethernet MAC, ARM processor, PCIe, …
  - Less flexible and number is limited, but very fast

Where to IP cores

- In IP management tools of design suites
  - Vivado has IP Integrator: supports both free and charged IP cores
- Downloadable RTLs: OpenCores.org and many other per-project sites
- Downloadable RTL generator: FloPoCo (Floating point tools) and similar tools

IP cores: pros and cons

- Pros: Designs made easier
  - Usually faster and/or smaller than hand-written HDL designs
- Cons: Designs made harder
  - IP core behavior must be considered in HDL-written modules
  - May be a problem when migrating to ASICs or other FPGAs

IP cores vs Designers

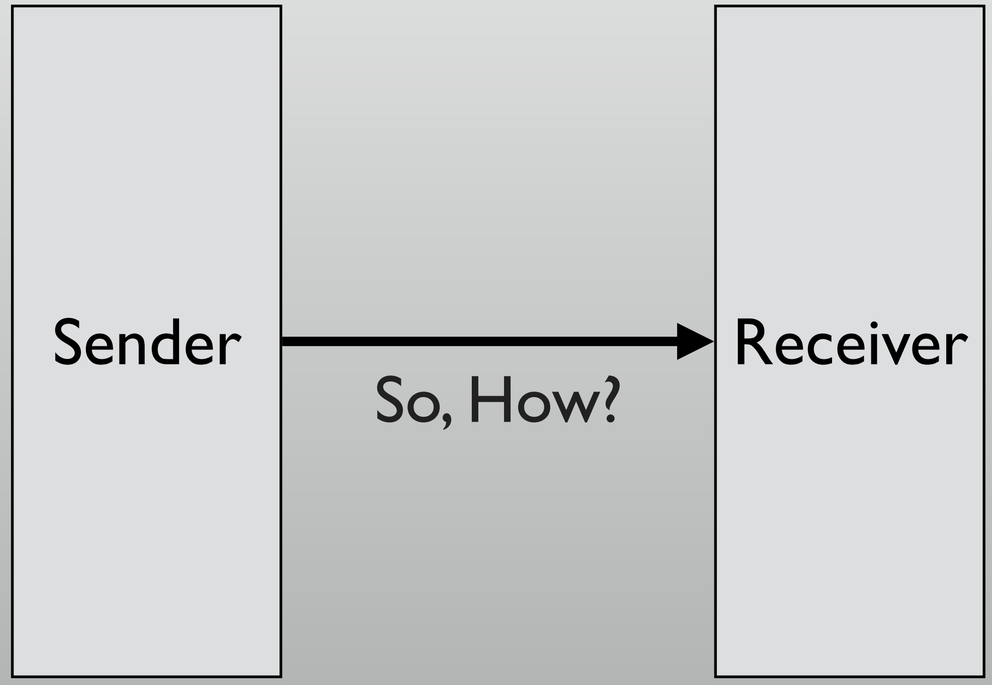
- Things to know
  - Interface, or module ports: How each port signals acts
    - Many IP cores have some standard compliant interface ports
  - Parameters: what is configurable in the IP core

Interface standards

- Connect IP cores directly, like drawing block diagram
  - Requires common interface → standards are defined
    - AXI4 (Xilinx / ARM), Avalon (Intel), Wishbone (OpenCores)
  - Block-diagram based design tools
    - IP Integrator (Xilinx), Qsys (Intel)

What is Interface standard?

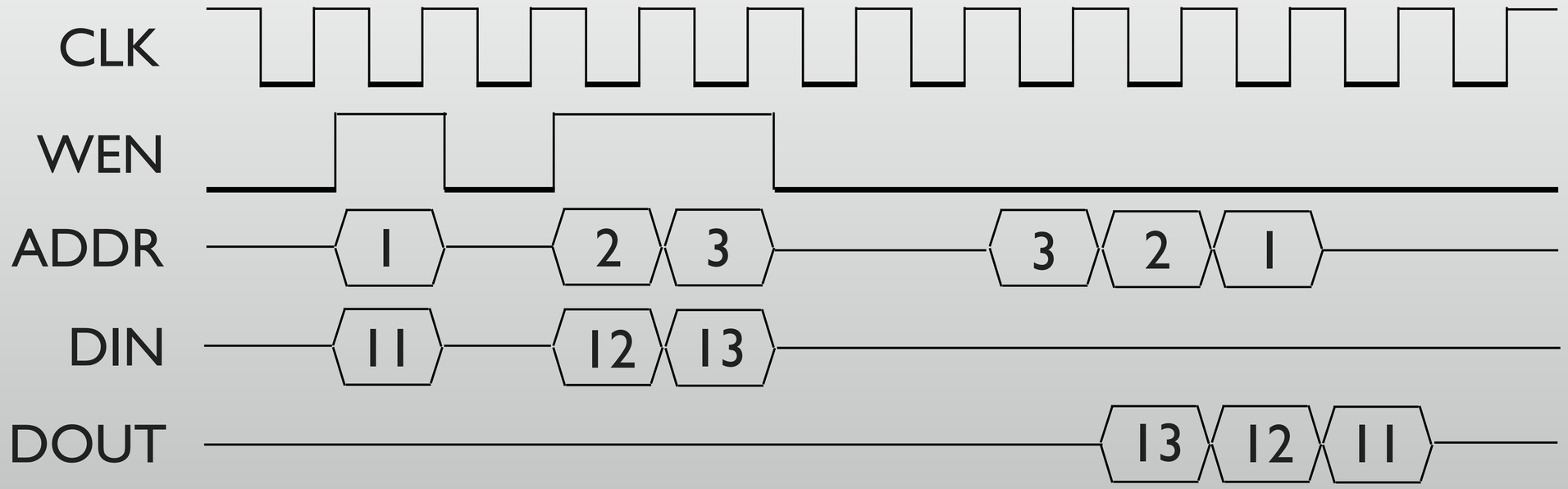
* How to connect sender and receiver
  * “Are you ready?” → “Yes / Wait!”
  * “Here data is” → “OK / Wait!”
  * “Over”



Well-used interface types

- Memory type
  - Access with address to read/write
  - Random access order is possible for both reads and writes
- Stream type
  - Data transfer in single direction
  - Read in written order only

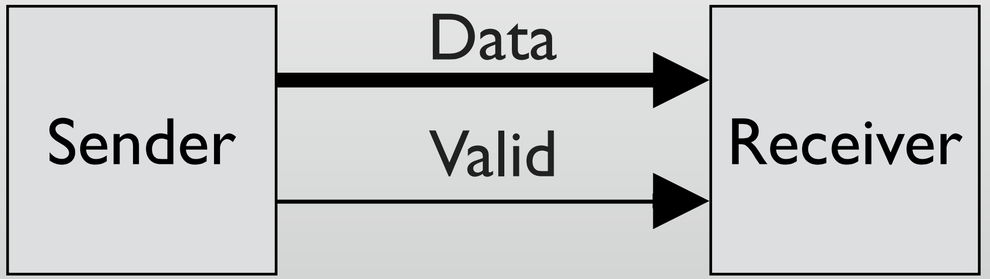
Block RAM interface

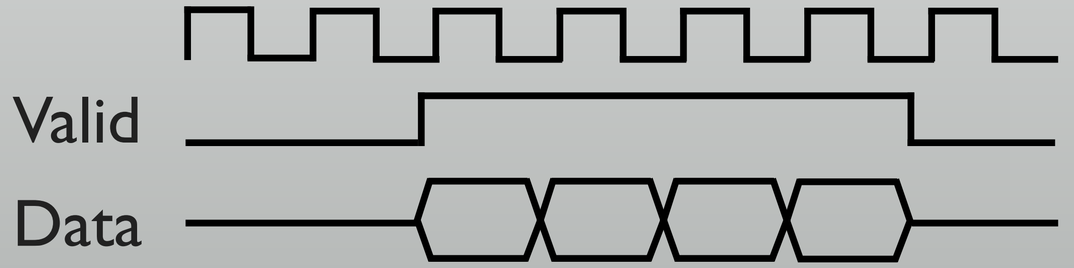


* Write data: same time with address
* Read data: 1 clock later than address

Simple stream transfer

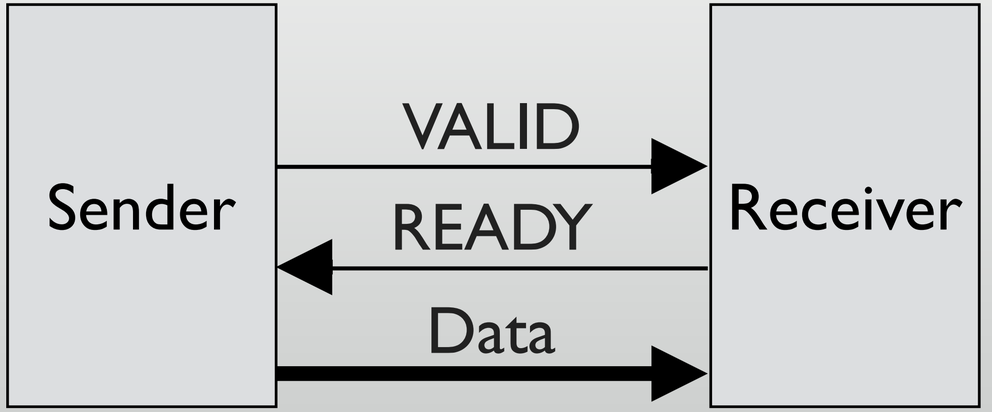
- Both synchronous to same clock
- Data is free-running
  - with validity signal

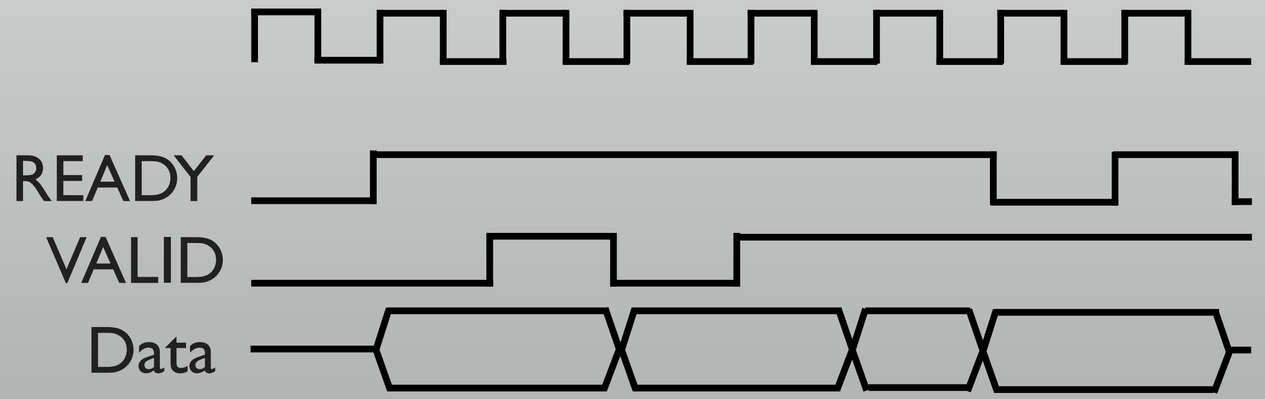




Sender/receiver handshake

- Check status each other
  - VALID + READY
  - Next data sent when both are ready to send/receive





Interface variations

- Both memory type and stream type has many options
  - AXI4 has both types and many options
  - Can be connected together, though
    - Even memory ←→ stream connection is possible with adapter IPs

Typical IP cores for FPGAs

- On-chip memory: BlockRAM or FIFOs
- Clock managers: to generate clock signals with different frequency or phase
- Floating-Point operators
- PCI Express, Ethernet and many other standard interfaces

Today’s exercise

- Using clock manager (CMT: Clock Management Tile)
  - LED flashing at 100MHz and 150MHz
- BlockRAM
  - Stopwatch version 2 with lap time feature

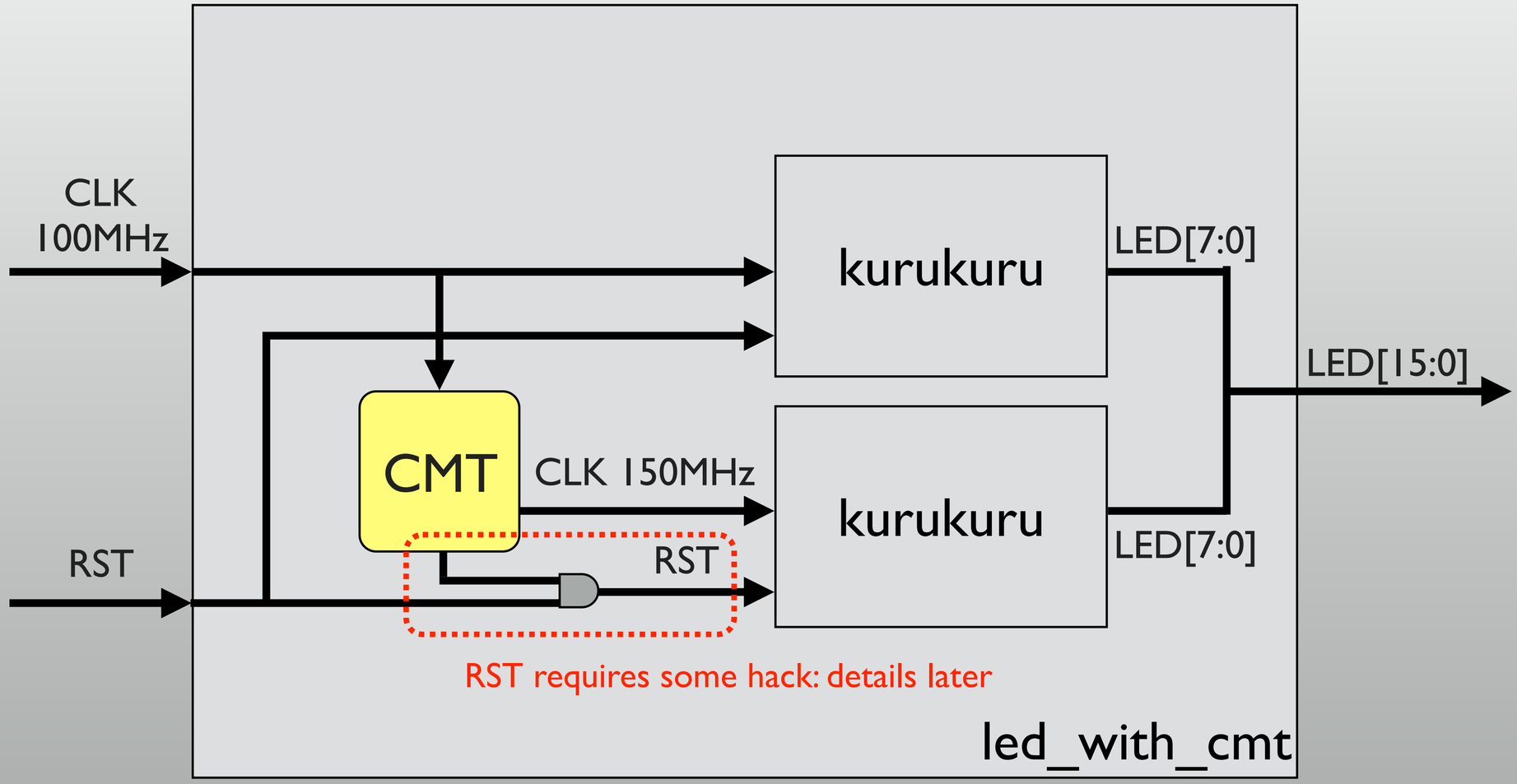
CMT (6 available on Artix-7 100)

- PLL (Phase Locked Loop) + DLL (Delay Locked Loop)
  - PLL uses VCO (voltage controlled oscillator), DLL uses delay lines instead
- Features:
  - Phase shift of 90°, 180° and 270°
  - Frequency generation by double, or with multipliers and dividers
    - Multiply by *n* / divide by *m* of input frequency

Block RAM (135 available on Artix-7 100)

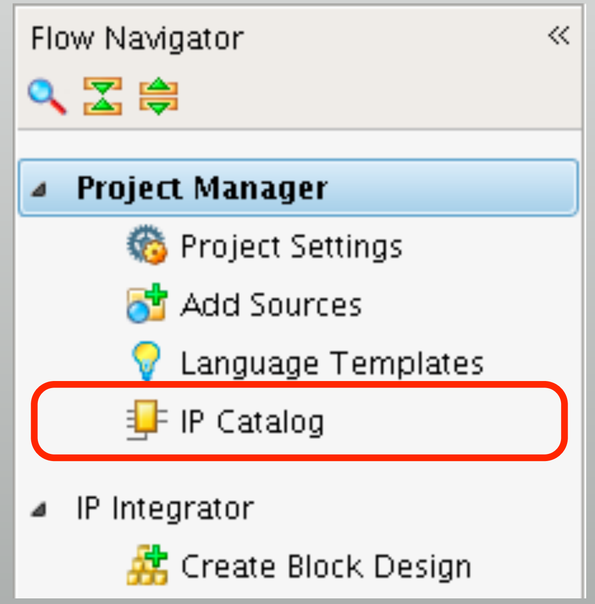
- 36kb dual port memory blocks, distributed all over the FPGA chip
  - Can be accessed individually or in parallel, provides a great bandwidth
  - Configurable width x depth of 1x32k, 2x16k, 4x8k, 9x4k, 18x2k, 36x1k
- Also available as 2 independent 18kb RAM blocks
- Or even as a large RAM by concatenating

Exercise1: LED flashing with CMT



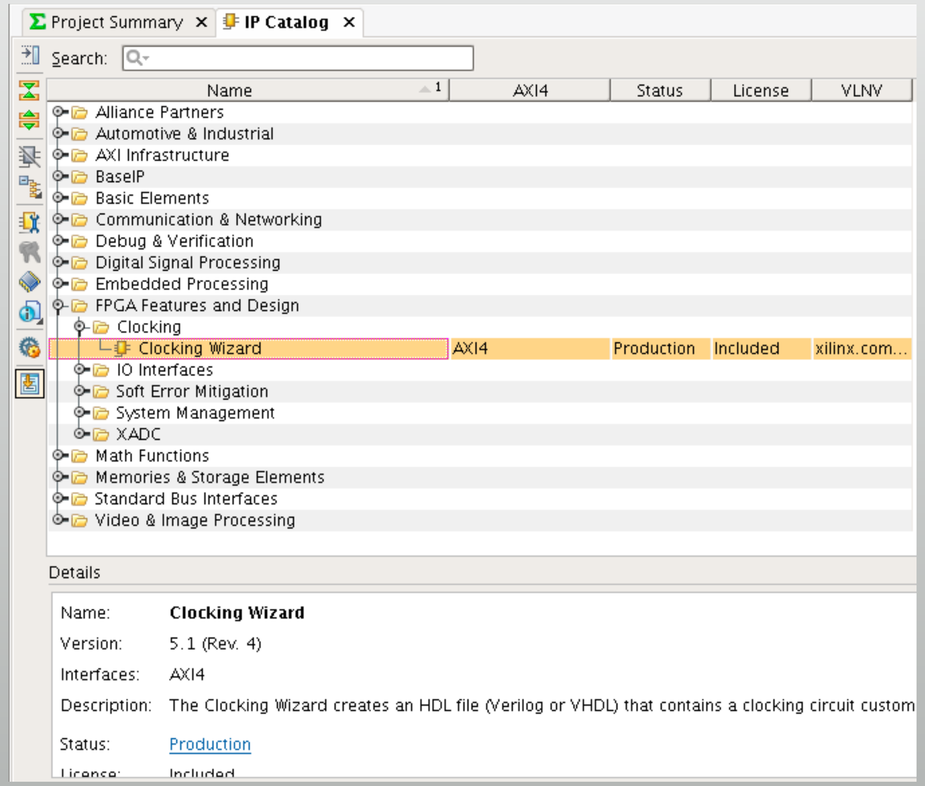
Exercise 1 HOWTO

- Design sources, constraints, testbench are in ip-lab/lab1/src
  - But there's no CMT
  - Open IP Catalog after all sources are set in project



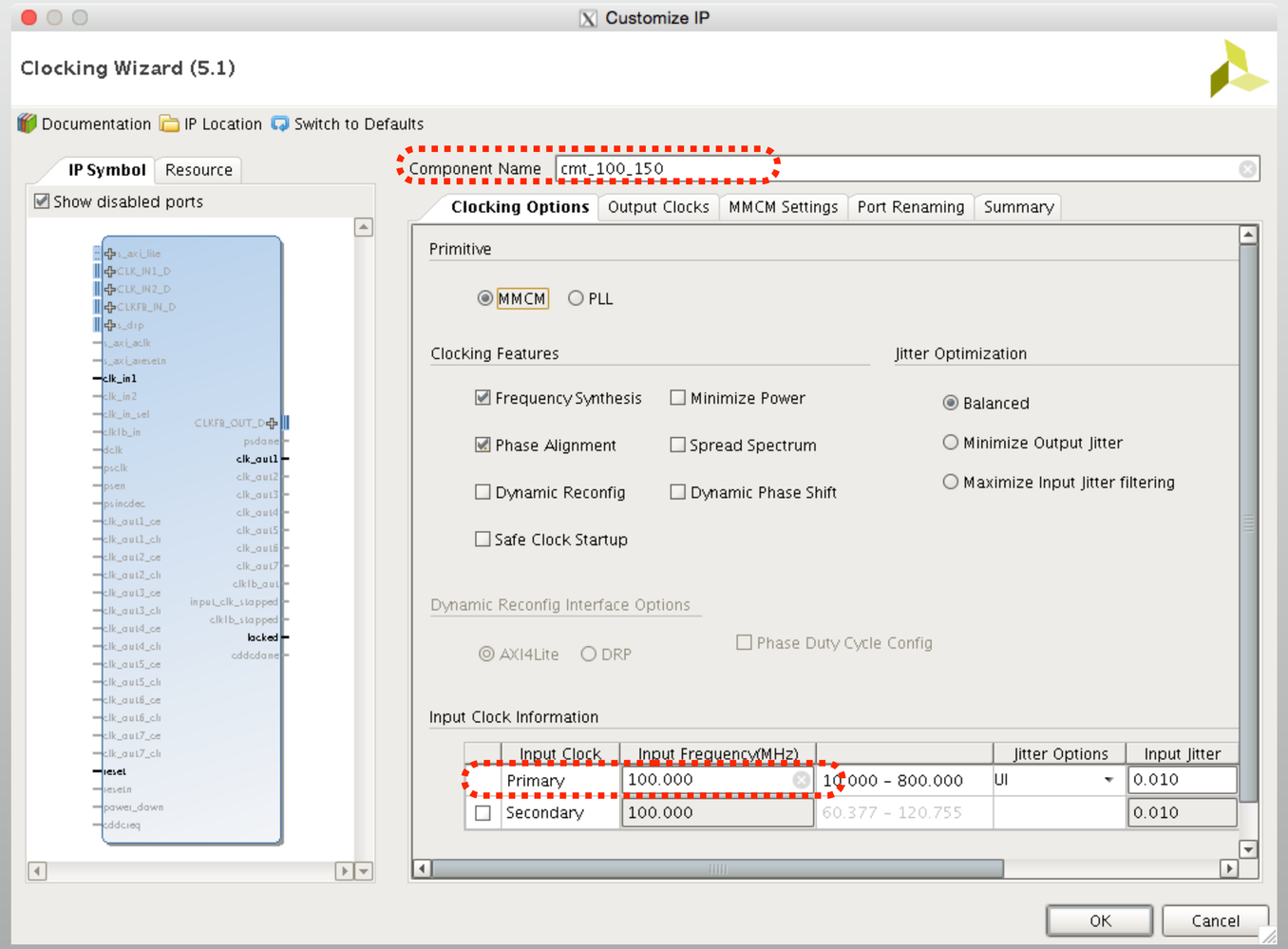
IP Catalog

- List of available IP cores
  - Launch Clocking Wizard to use CMT
    - FPGA Features & Design → Clocking → Clocking Wizard



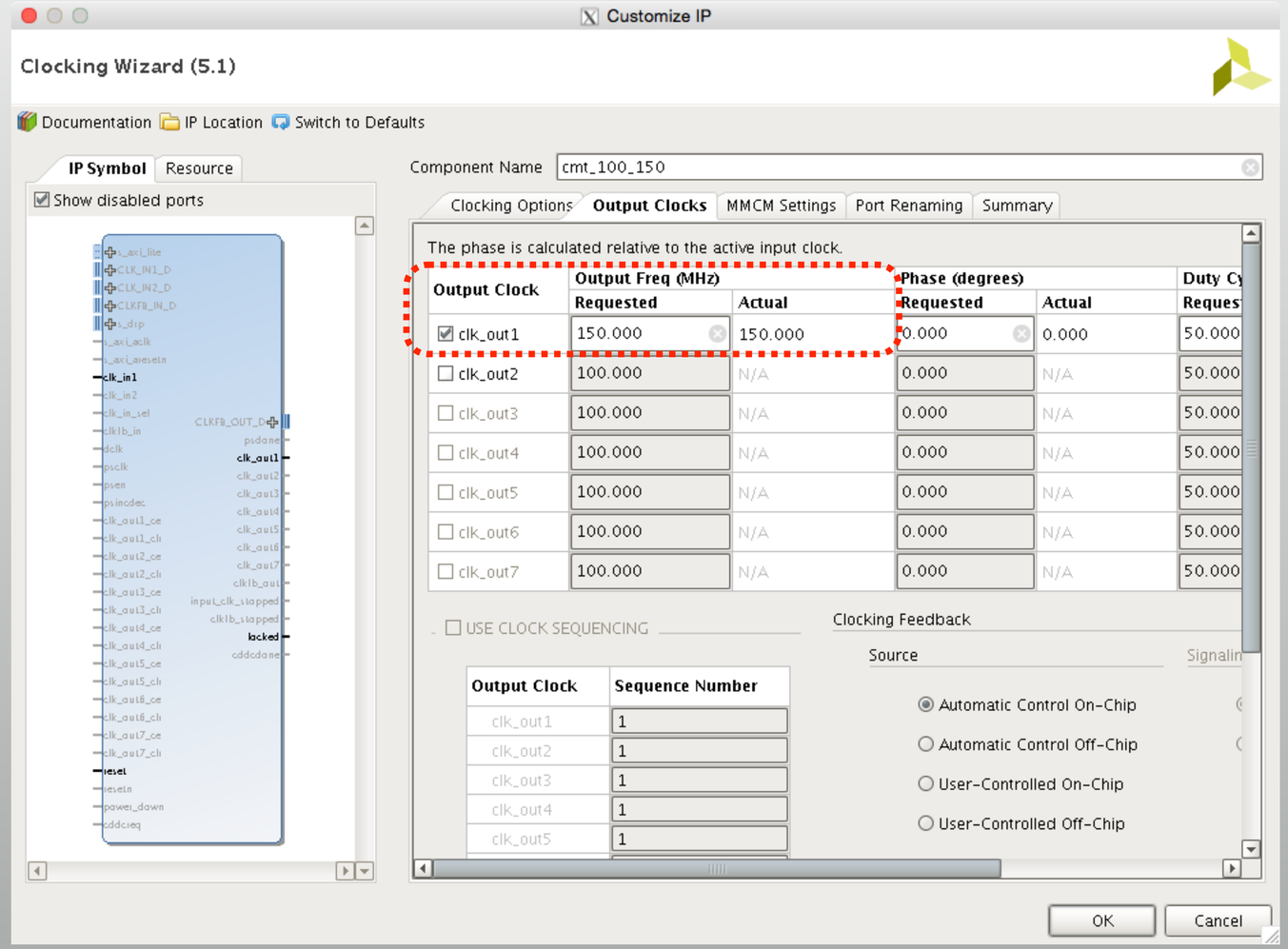
Clocking Wizard (1/2)

- Set the module name
  - Default isn't good to avoid unexpected overwrite…!
- Set the input frequency (100MHz)



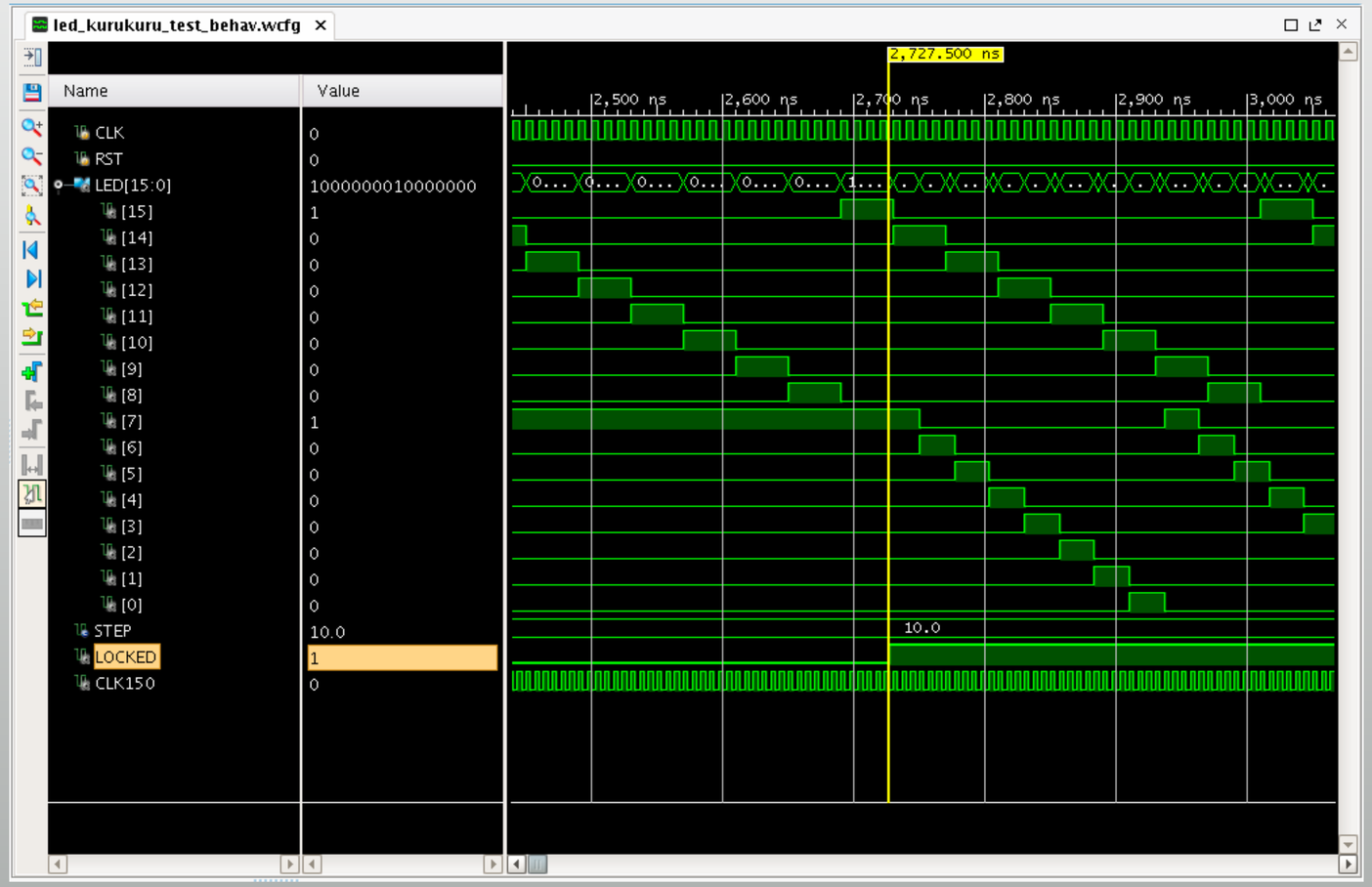
Clocking Wizard (2/2)

- Set output frequency
  - “Requested” and “Actual” are sometimes different
  - Because multiplier / divider has some restriction
    - ex) Try 123.0MHz



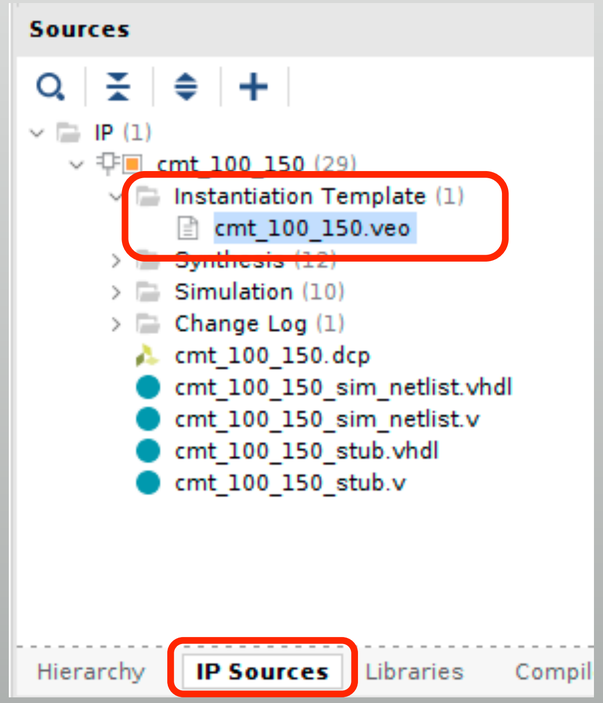
Running simulation

- DLL locks at 2.7us
  - LOCKED goes high
  - This must be added to RST condition
  - Check source code for detail



To instantiate generate core…

- The module's port definition is necessary
  - Opening the generator GUI is not very effective
  - Find the core in "IP sources", then there's "Implementation template "
    - .veo is the template for Verilog



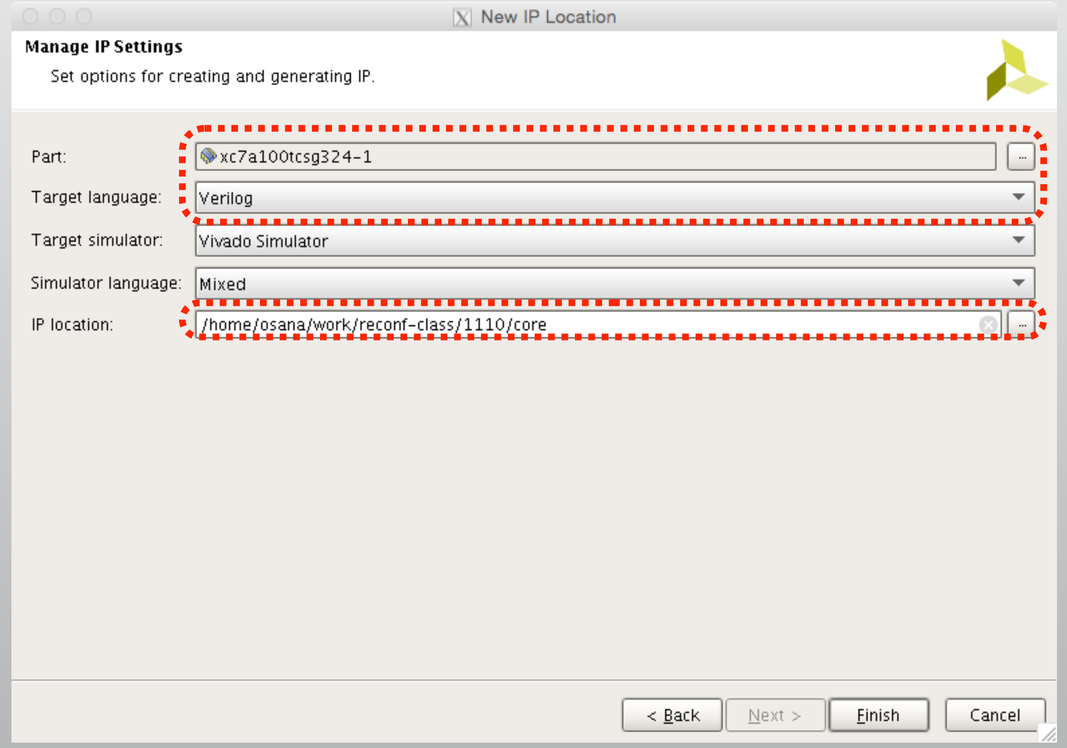
Where’s generated cores?

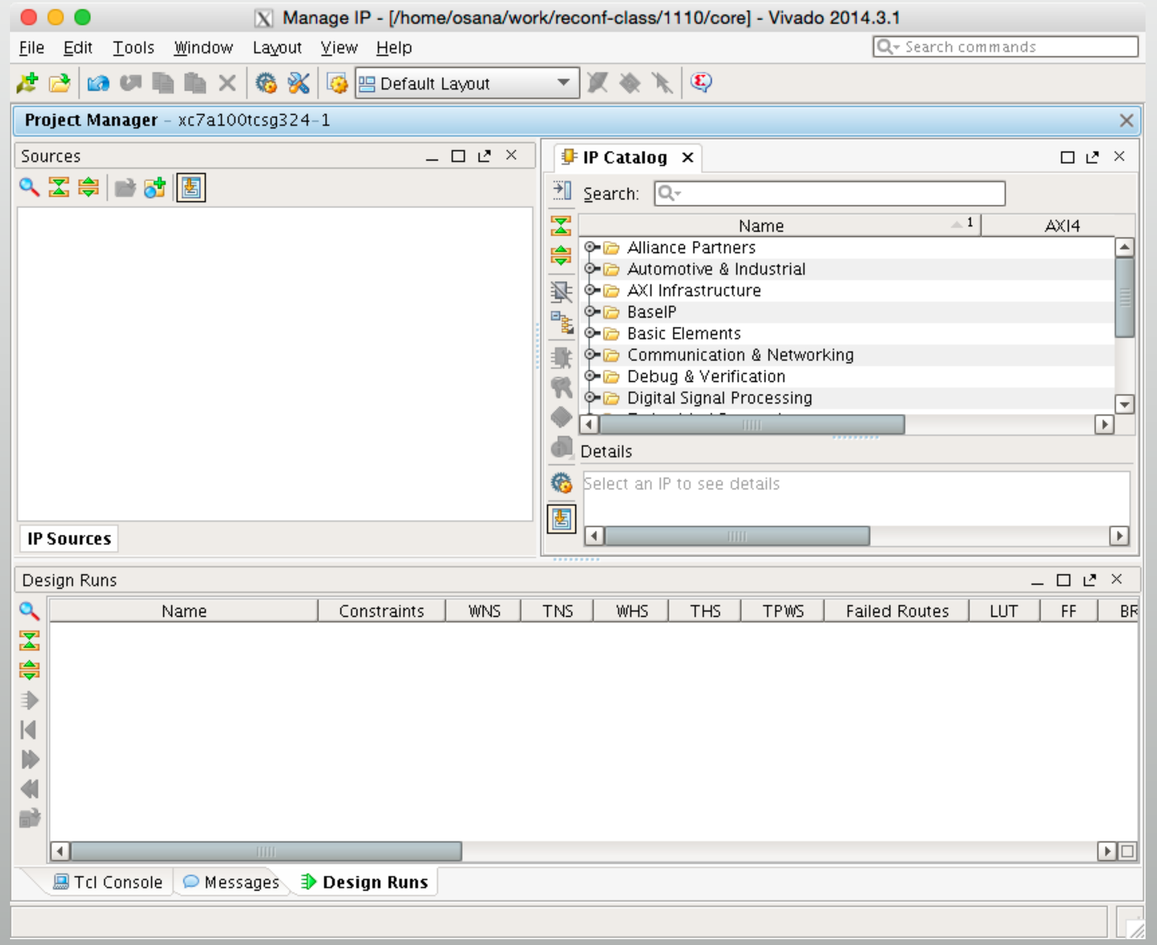
- project_1/project_1.srcs/sources_1/ip/cmt_100_150/
  - Enough good for exercise or one-time test project
  - Not good when migrating a test to large, complete project
    - For large projects, managing IP cores separately is good

IP Location: manage cores outside

- "Manage IP" in Vivado startup screen
  - Make a IP Location to manage generated IP cores outside an RTL project

Looks very similar to RTL project





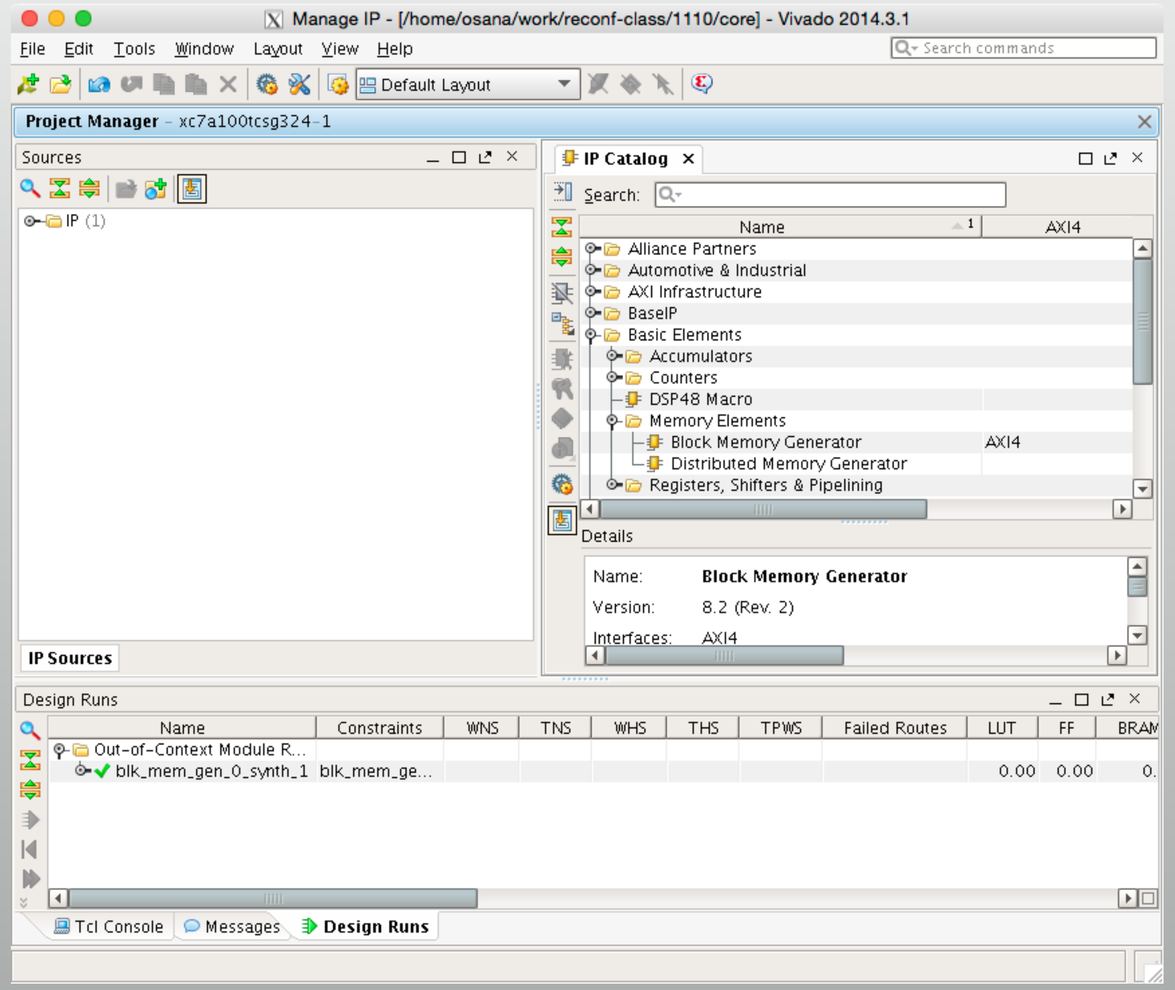
To use the core, add generated ".xci" or ".xcix" files to RTL project

Exercise 2: Using Block RAMs

- Make an IP location
  - lab2/core
- Generate a Block RAM (32bit x 1024・True dual port)

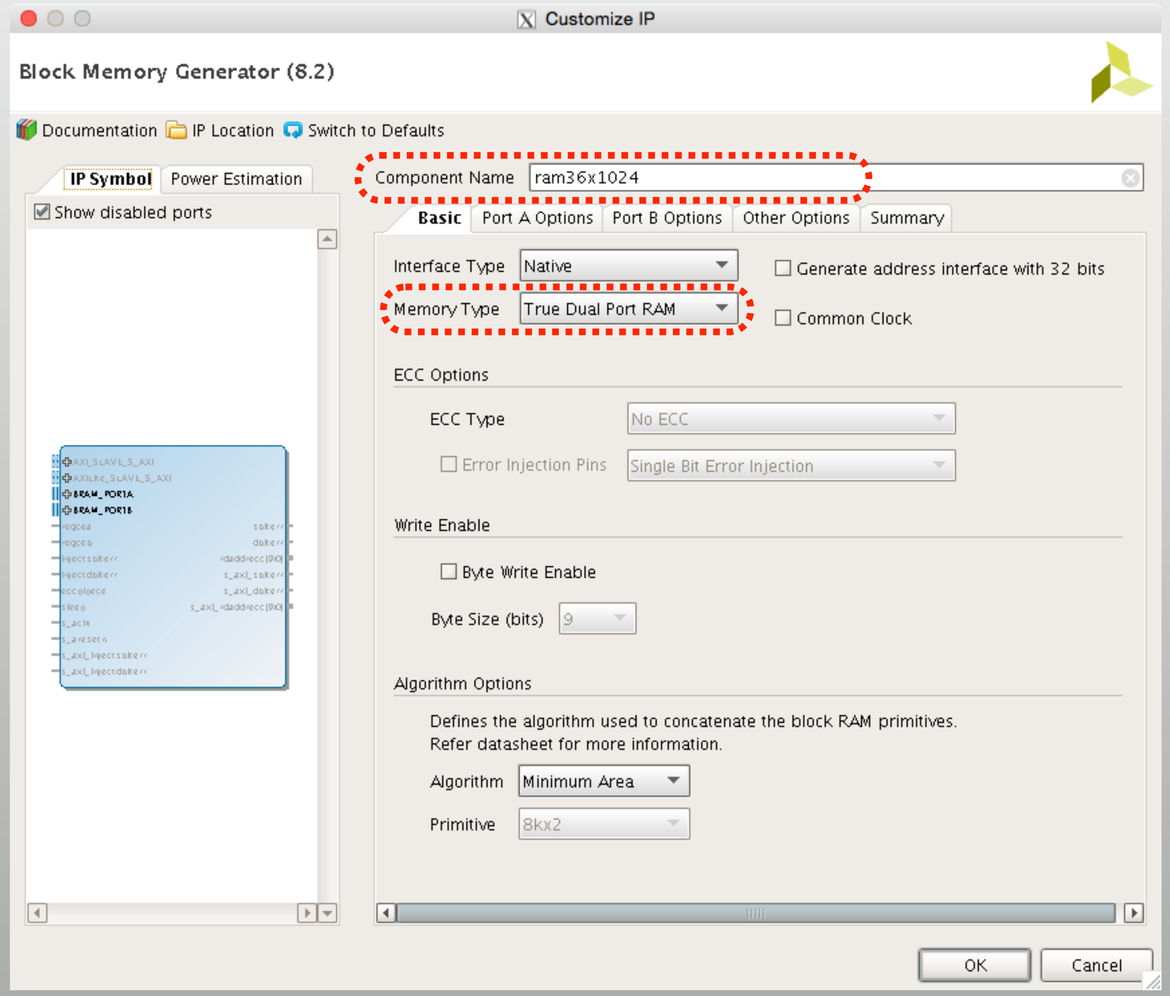
Block Memory Generator

- Basic Elements → Memory Elements
  - Block and distributed memory
    - Hard IP (Block) or LUTs (distributed)
  - Block Memory for today



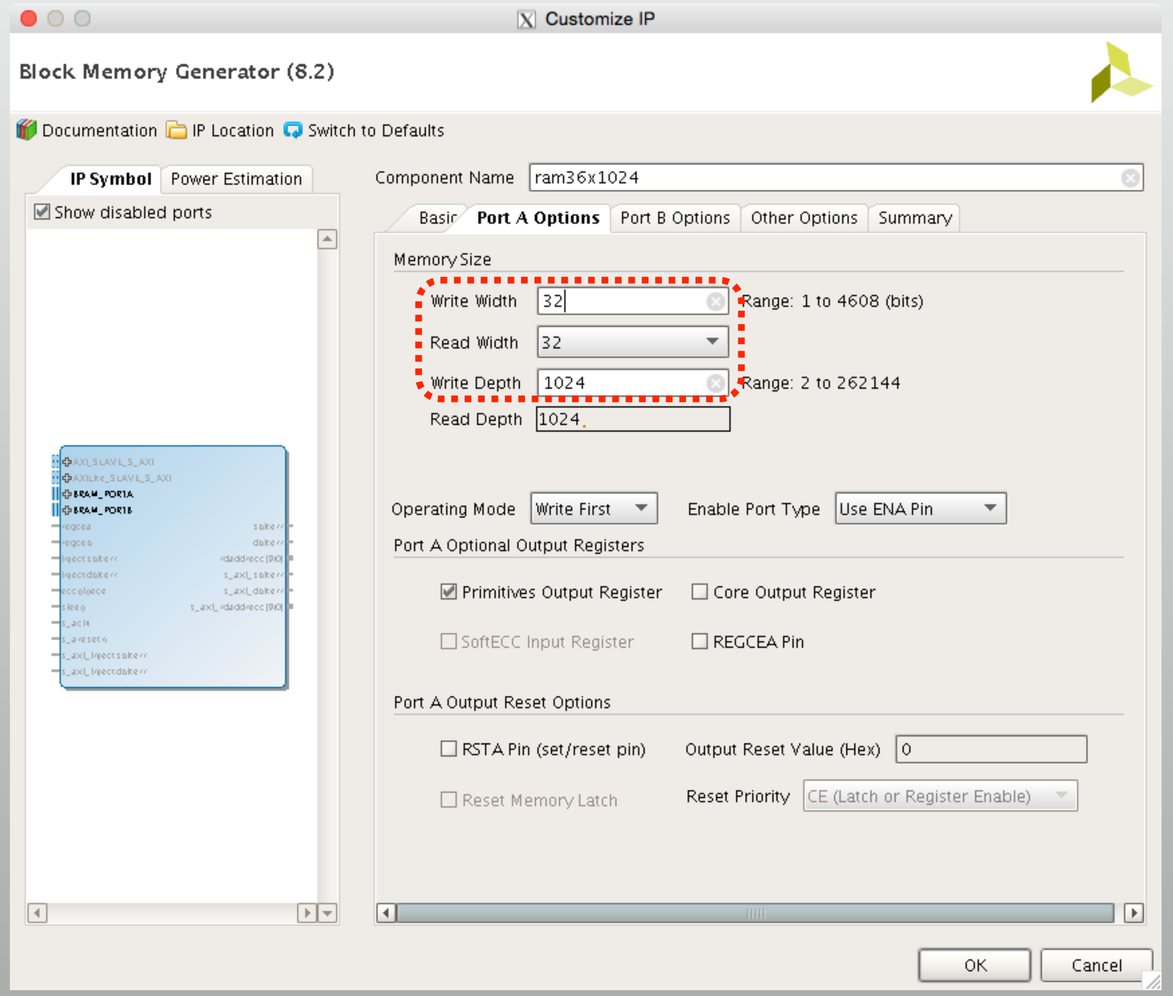
Basic

- Component Name: ram36x1024
- Memory Type: True Dual Port RAM



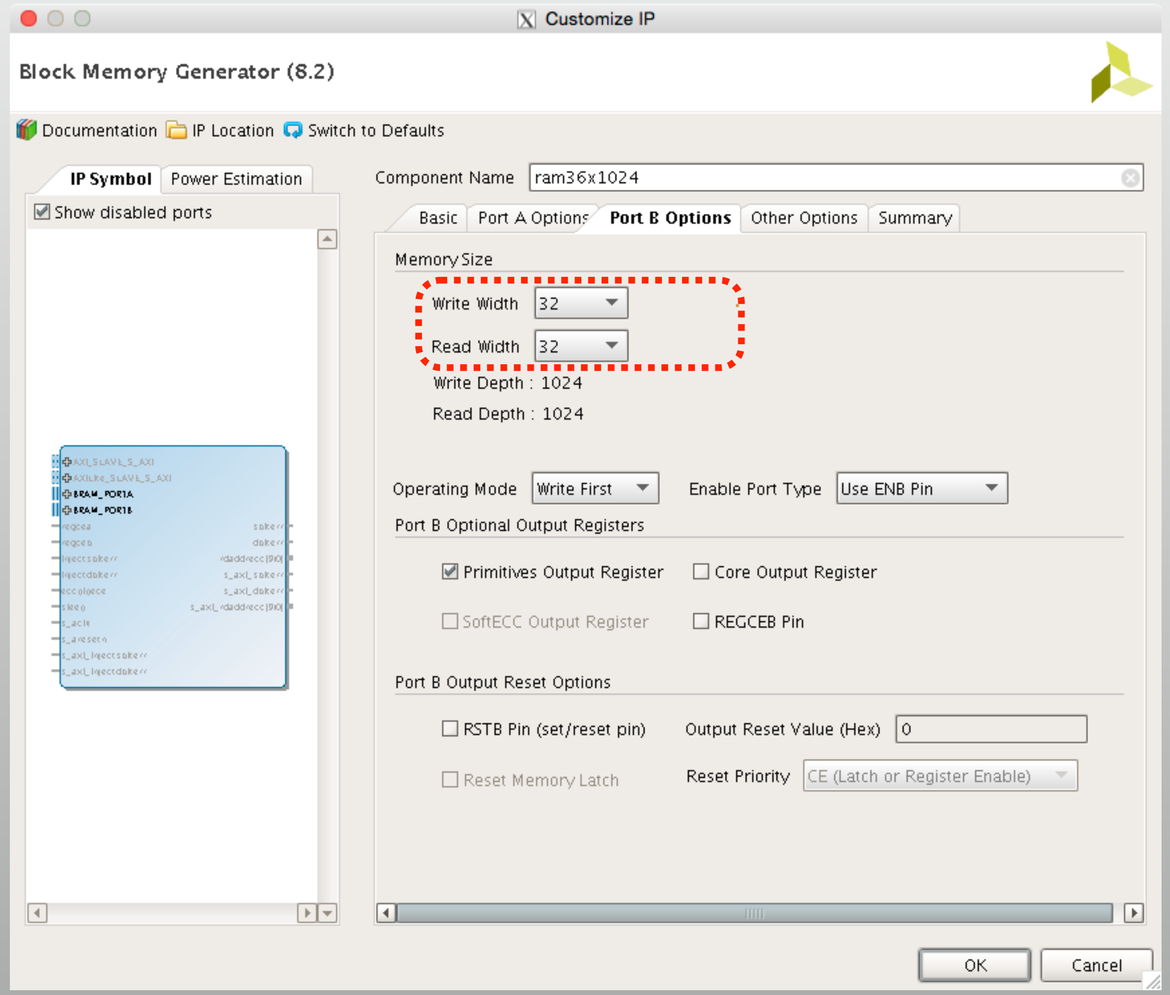
Port A Options

- Read / Write Width: 32
- Write Depth: 1024



Port B Options

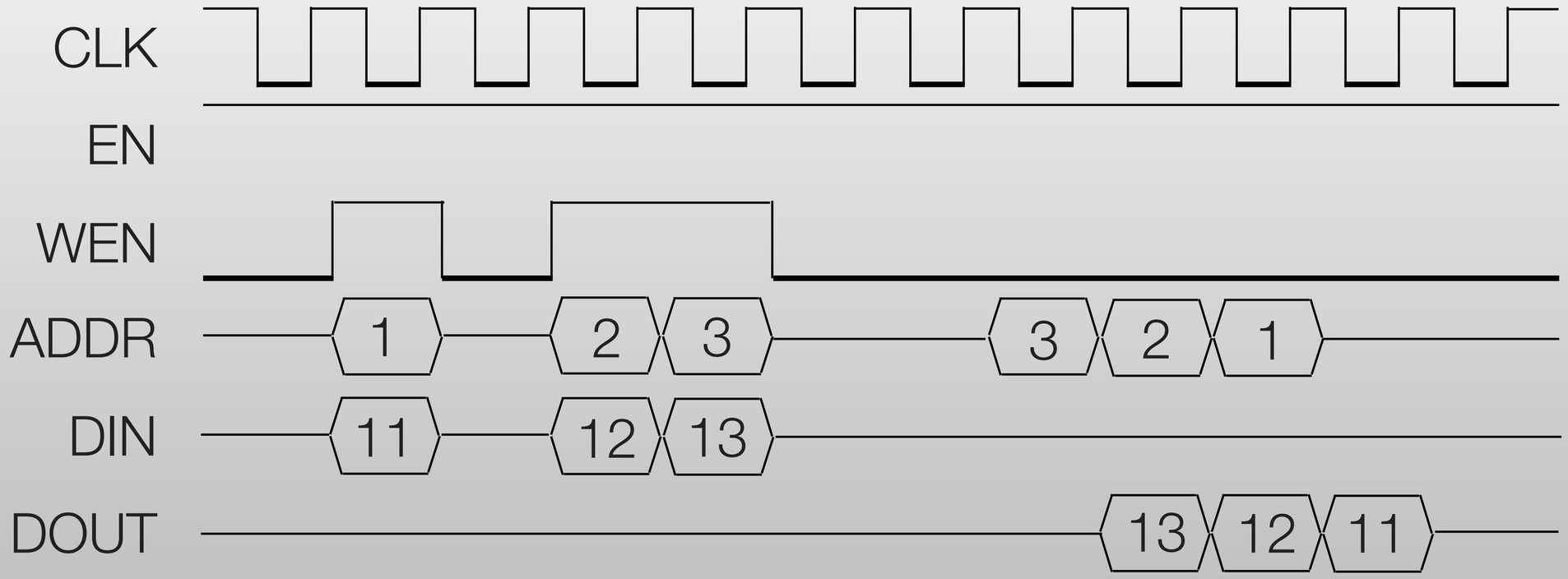
- OK if Read/Write are 32bit



Assignment #2

- 12/3: deadline for stopwatch
  - Start/Stop and Reset button
- 12/17: deadline for stopwatch 2.0
  - Start/Stop, Reset and Lap
  - Up to 1000 laps, any usage / funtions of buttons and switches are OK

おまけ: Block RAM のアクセス

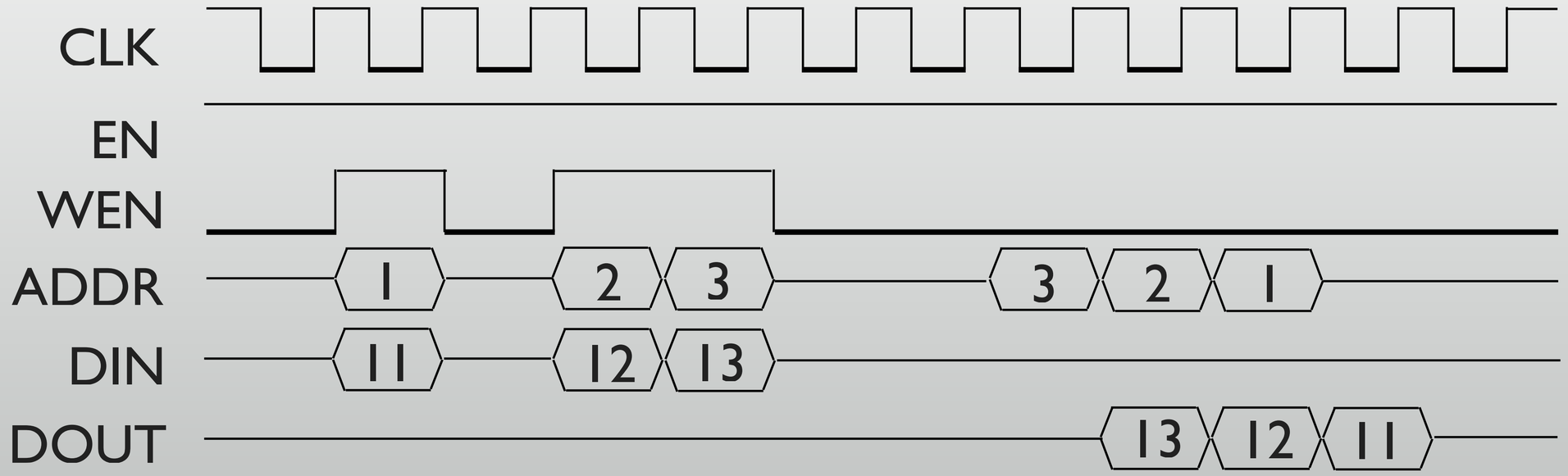


書き込みは ADDR + WEN + DIN が同時、

読み出しは DOUT が1サイクル遅れます。

DOUT は実際にはずっと何かしら出ています。

One more thing on Block RAM



* “EN (enable)” must kept high while Block RAM is active
* Actually, DOUT is also active while the RAM is being written