HECTOR

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052

HECTOR Evaluation Platform

Tutorial

O. PEŤURA[1], M. LABAN[1,2], V. FISCHER[1]

[1]Univ Lyon, UJM – Hubert Curien Laboratory
[2]MICRONIC a.s.

Objectives of this tutorial 1/2

- Present HECTOR evaluation platform
  - Hardware
  - Firmware
  - Software
- Present reference designs of the platform
  - Motherboard only
    - ≫ Controlling outputs of the motherboard
    - ≫ Processing a data file (input → processing → output)
  - Motherboard & Daughter board
    - ≫ Reading counter values from the Daughter boards (M, A)
    - ≫ Reading random data from the Daughter boards (M, A)
      - Plugged to the SATA connector
      - Connected via the HDMI cable
- Let you make your own design

Objectives of this tutorial 2/2

- To share our knowledge
- To simplify your developments
- To ask you to contribute on several levels
  - System level
  - Hardware level
  - Firmware level
  - Software level

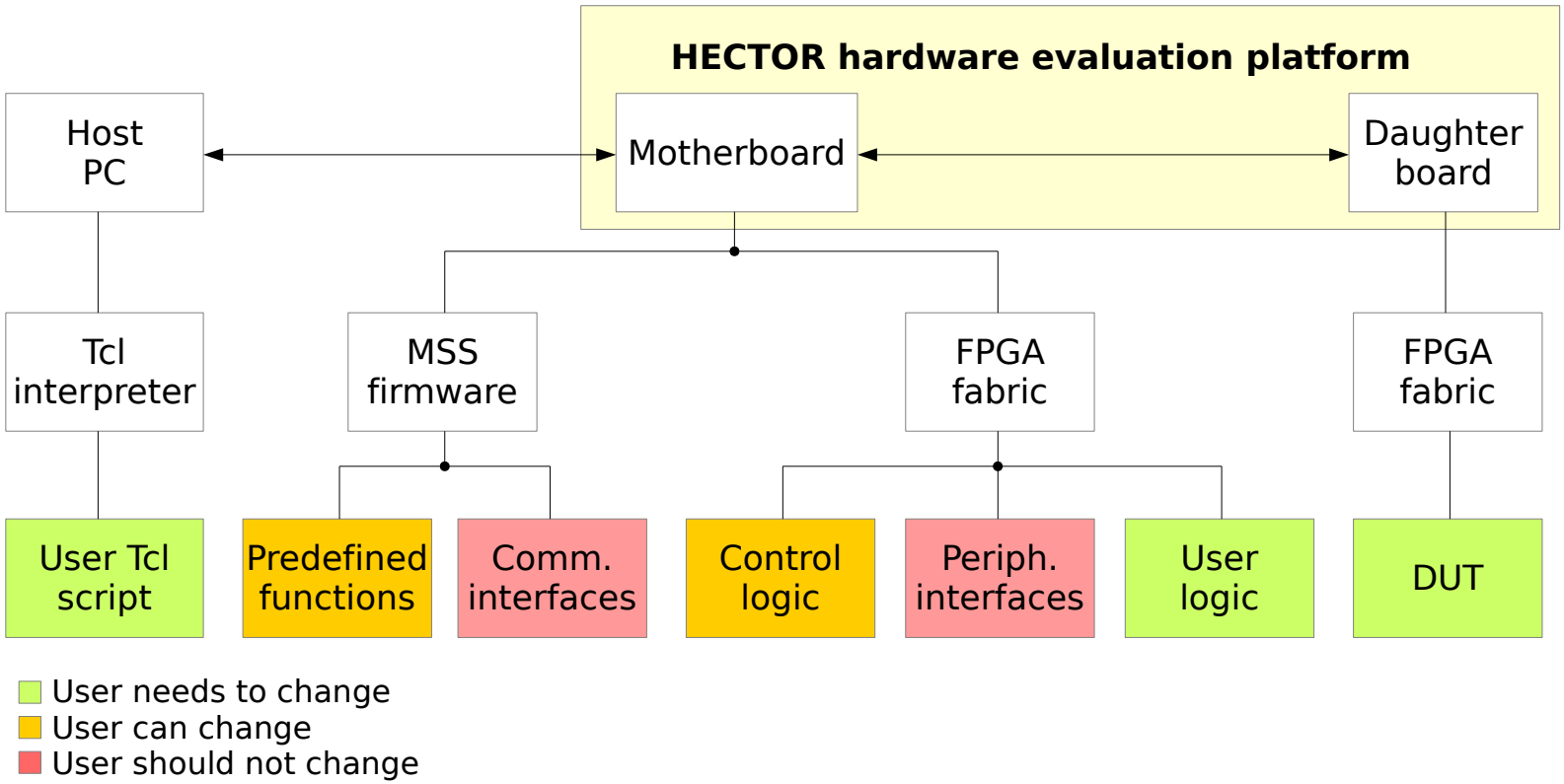
Outline

HECTOR – Hardware enabled crypto and randomness

- New approaches in the hardware design of
  - Random number generators
  - Physical unclonable functions
  - SCA-hardened authenticated encryption
- Evaluation platform
  - Suitable for implementation of TRNGs, PUFs and side channel attacks
  - Flexible and easy to use (and attack?)
  - Architecture close to that of the demonstrators
- Demonstrators
  - Robust – difficult to attack
  - Should validate/illustrate the main results of the project

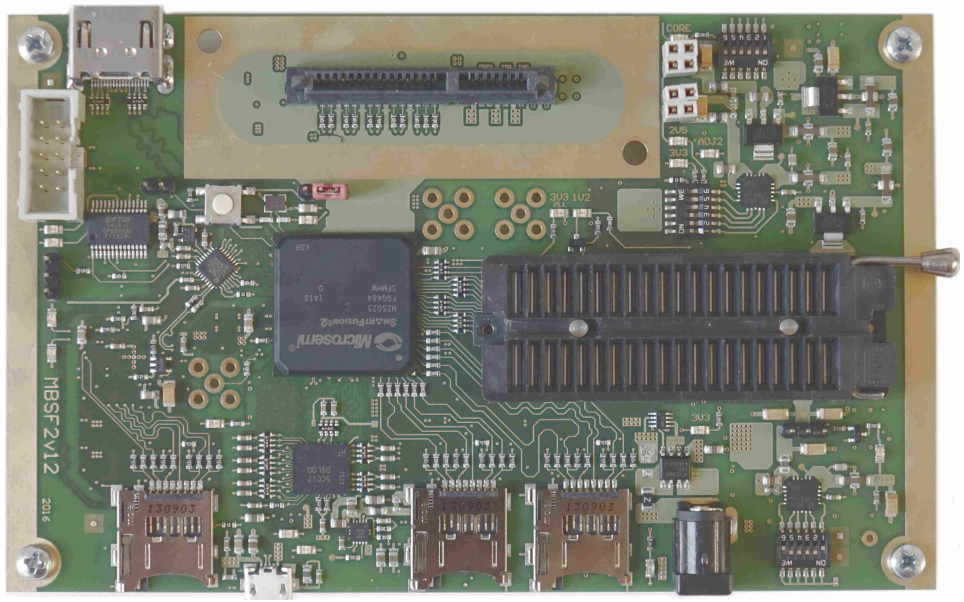
HECTOR evaluation platform toolkit – overview

- Sofware
- Firmware
- Hardware



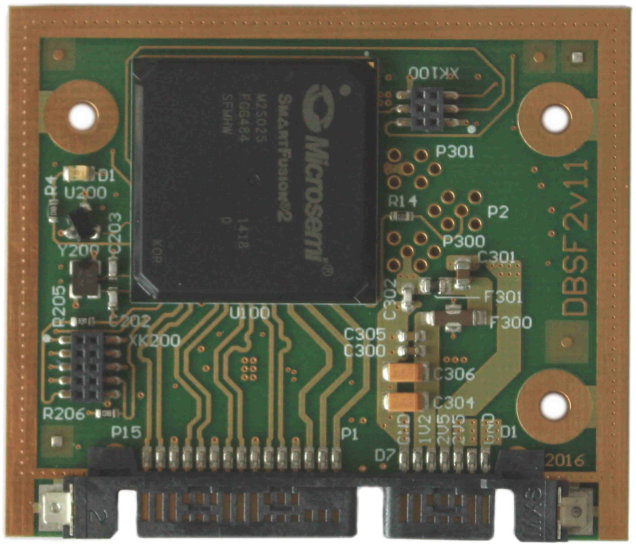
HECTOR evaluation platform – motherboard

- Basic element of the evaluation platform
- Provides data acquisition facilities
- Provides means of communication between PC and design under test (DUT)



HECTOR evaluation platform – daughter boards

- Primary platforms for DUT implementation
- Small, simple, and cheap design (for PUFs)
- Minimize external interference
- Can be connected via HDMI cable
- Three daughter boards available
  - Microsemi SmartFusion 2
  - Altera Cyclone V
  - Xilinx Spartan 6



Intended applications

- Evaluation of cryptographic primitives
  - TRNGs
  - PUFs
  - Authenicated encryption
- Comparison across different FPGA families
- Evaluation of encryption schemes
- Implementation of a complete cryptographic system (prototype of the demonstrator)
- Implementation of side channel attacks

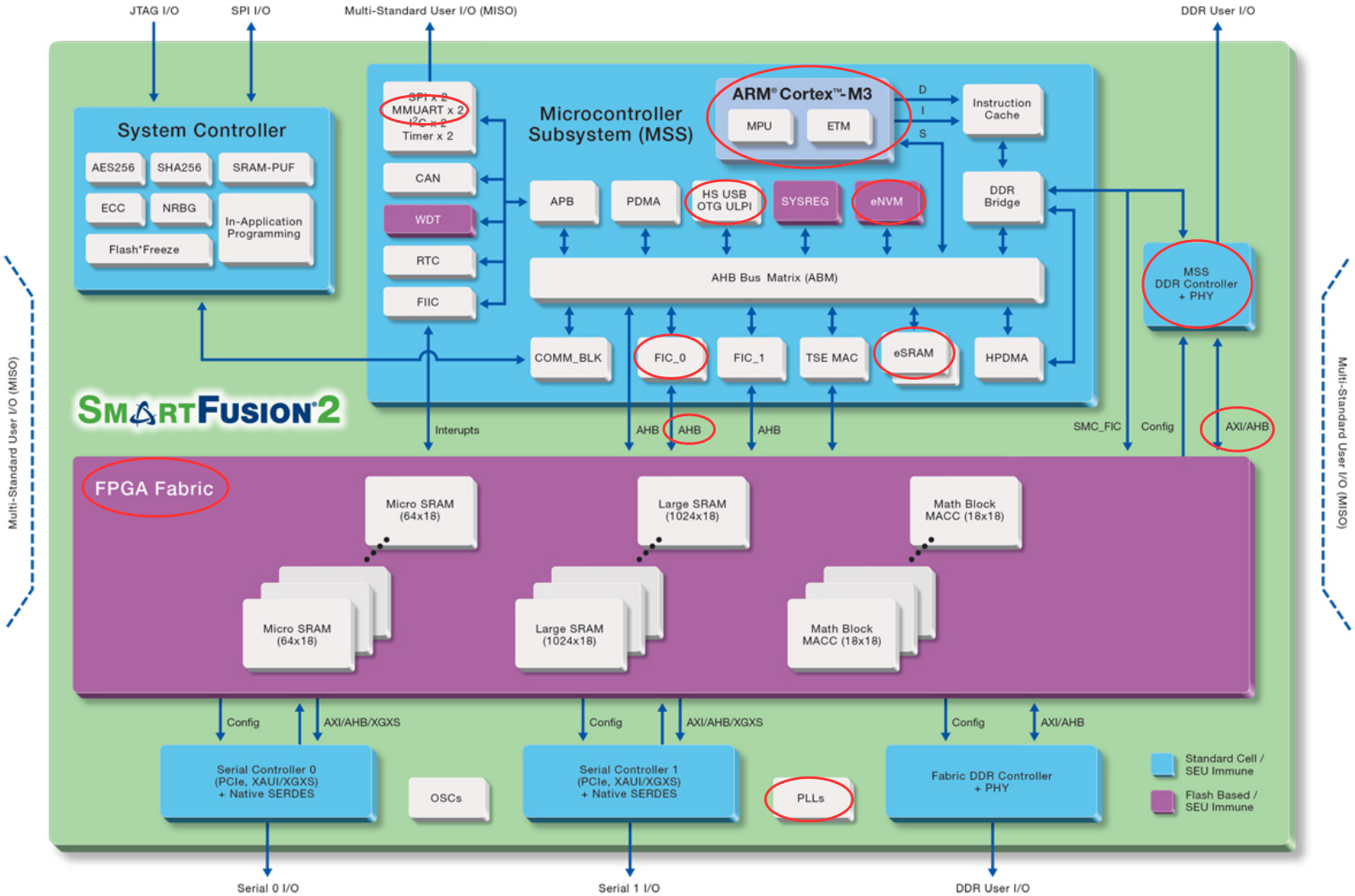
Software – on the Host PC

- Controls the data acquisition system via a serial link (USB $\rightarrow$ UART) using Tool Command Language (TCL)
- Can access DDR RAM on the HECTOR board via a mass-storage USB driver to send/receive large files (up to 30 MB)

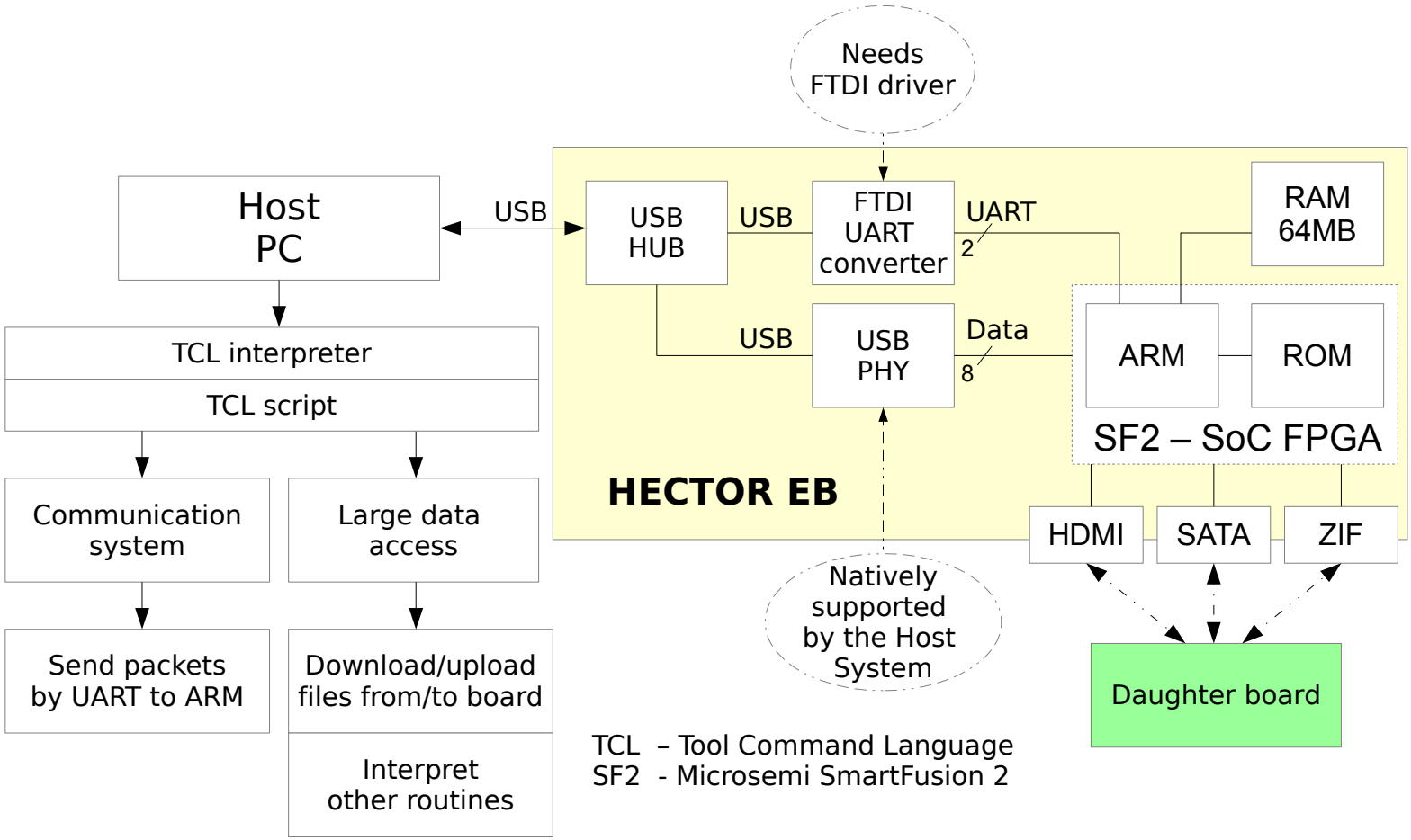
Motherboard

- FPGA SoC
  - ARM microcontroller
    - ≫ Receives commands from PC
    - ≫ Controls the DUT by sending commands to the FPGA fabric
    - ≫ Maintains a filesystem for data transfers (up to 32 MB in size)
  - FPGA Fabric
    - ≫ Provides DMA access to the DDR RAM for the DUT
    - ≫ Serves as a communication channel between the PC and DUT
- USB hub connected to
  - USB $\rightarrow$ UART converter
  - ARM-created filesystem on the DDR RAM
  - One SD card reader
- Three daughter board connectors (requested by HECTOR partners)
- Three micro-SD card sockets + 64 MB DDR SDRAM

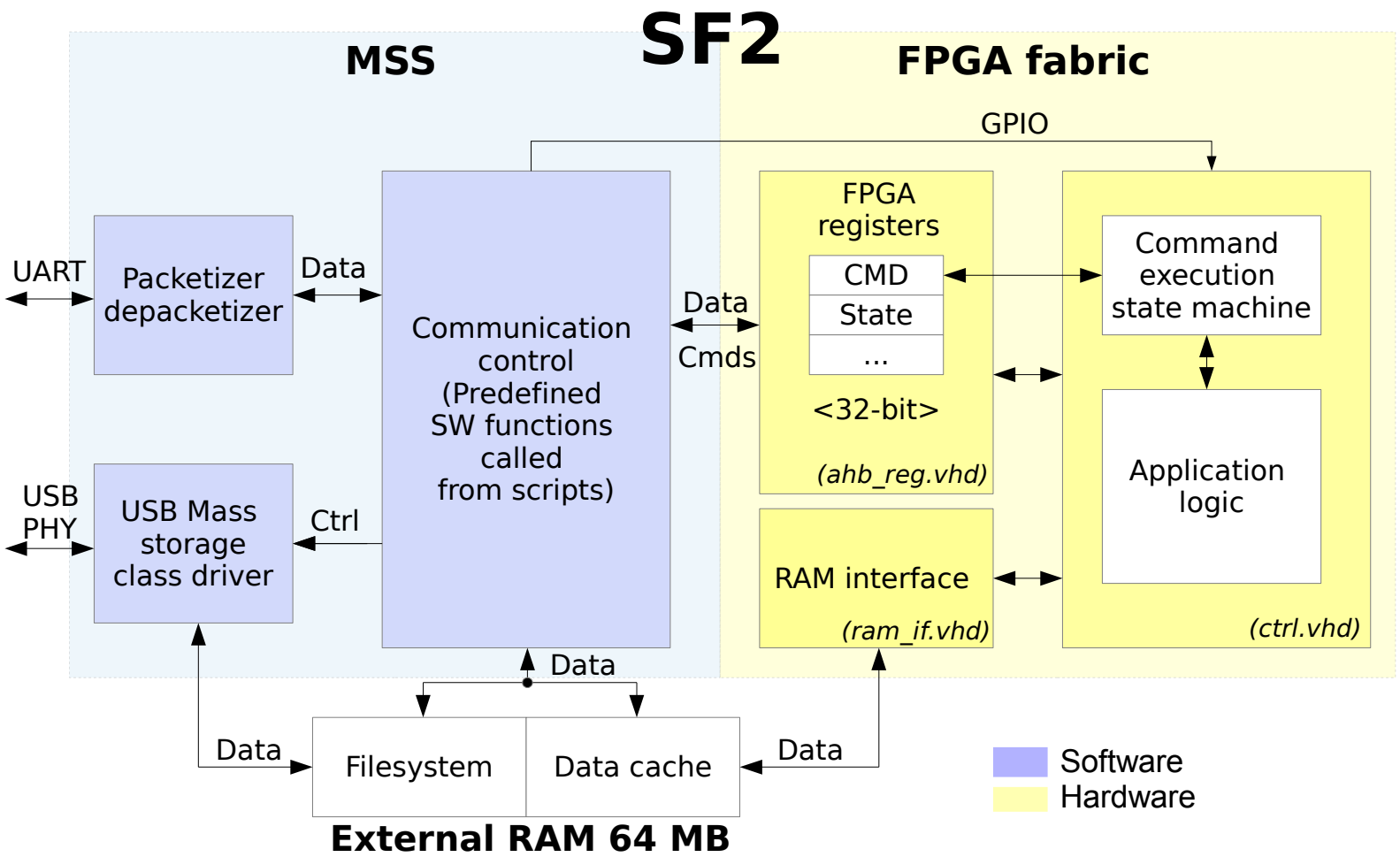
Microsemi SmartFusion2 SoC



Communications within the evaluation platform

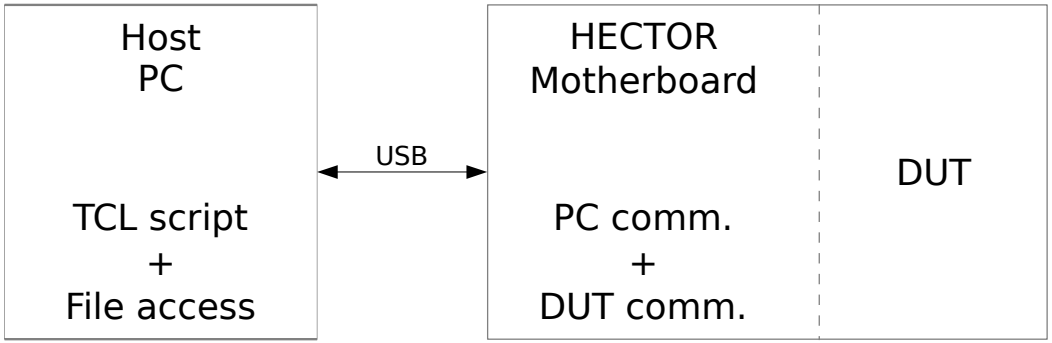


Functional diagram of the motherboard



Design model 1

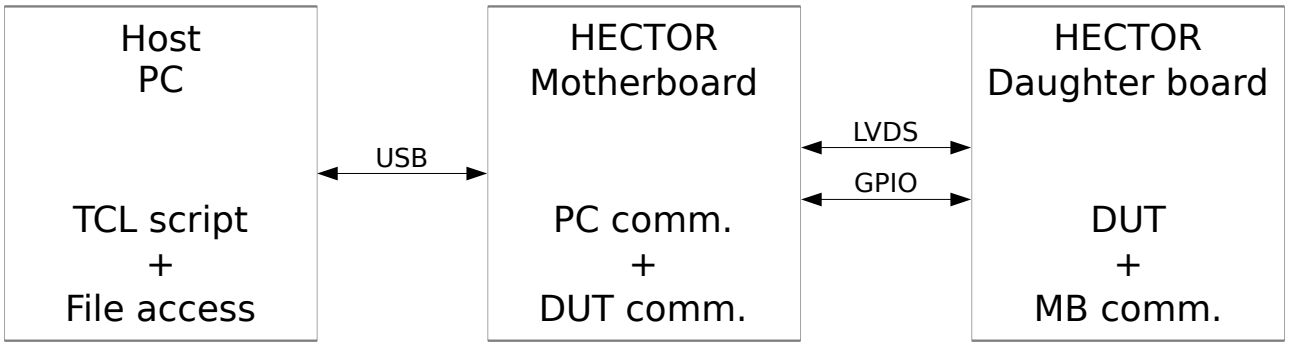
DUT implemented directly in the motherboard



- Better for development
- More flexible & larger data interface
- Better suited for implementation of cryptographic algorithms

Design model 2

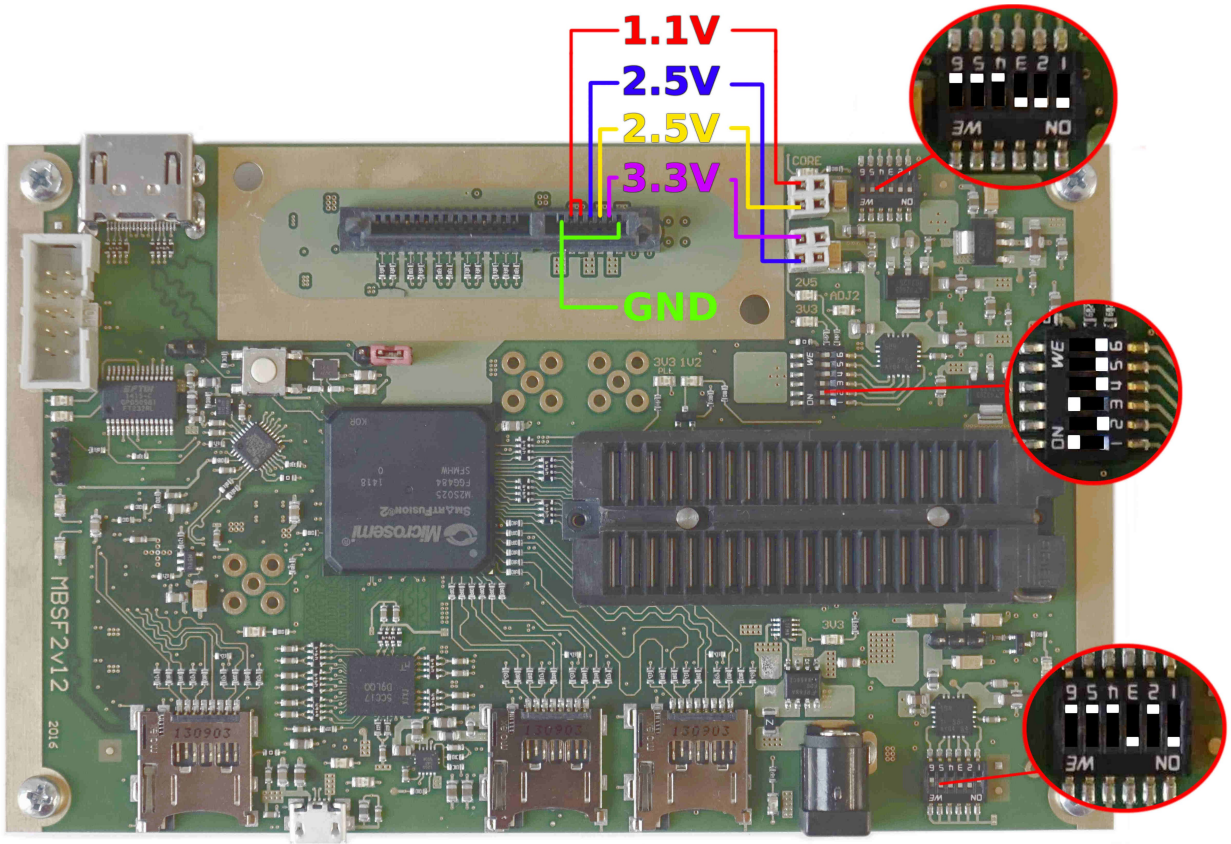
DUT implemented in the daughter board



- Well suited for testing TRNG and PUF implementations
- DUT can be tested remotely in a controlled environment:
  - Temperature – temperature controlled chamber
  - Electromagnetic interference – Faraday cage
  - Voltage – external power supplies

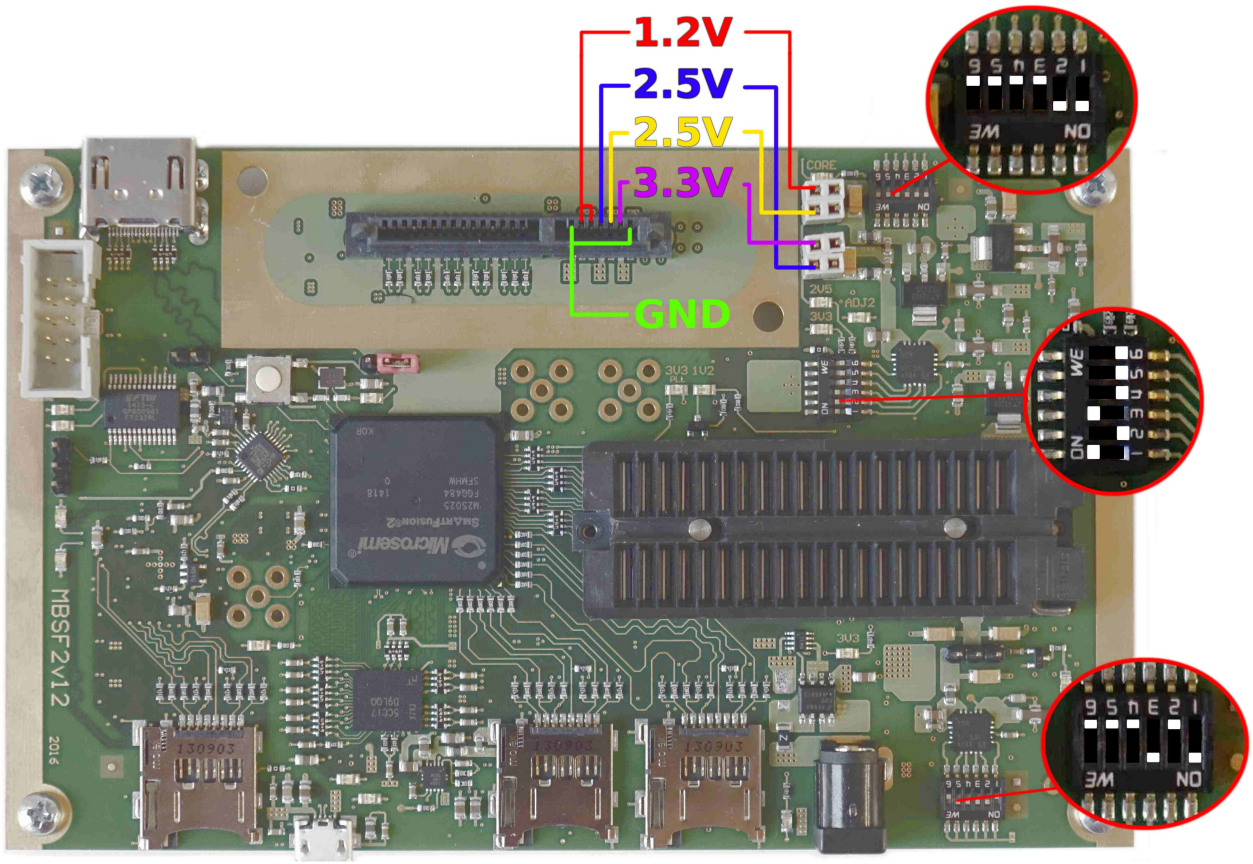
Setting-up power supplies for daughter boards (1/2)

Configuration for daughter board
Altera Cyclone V

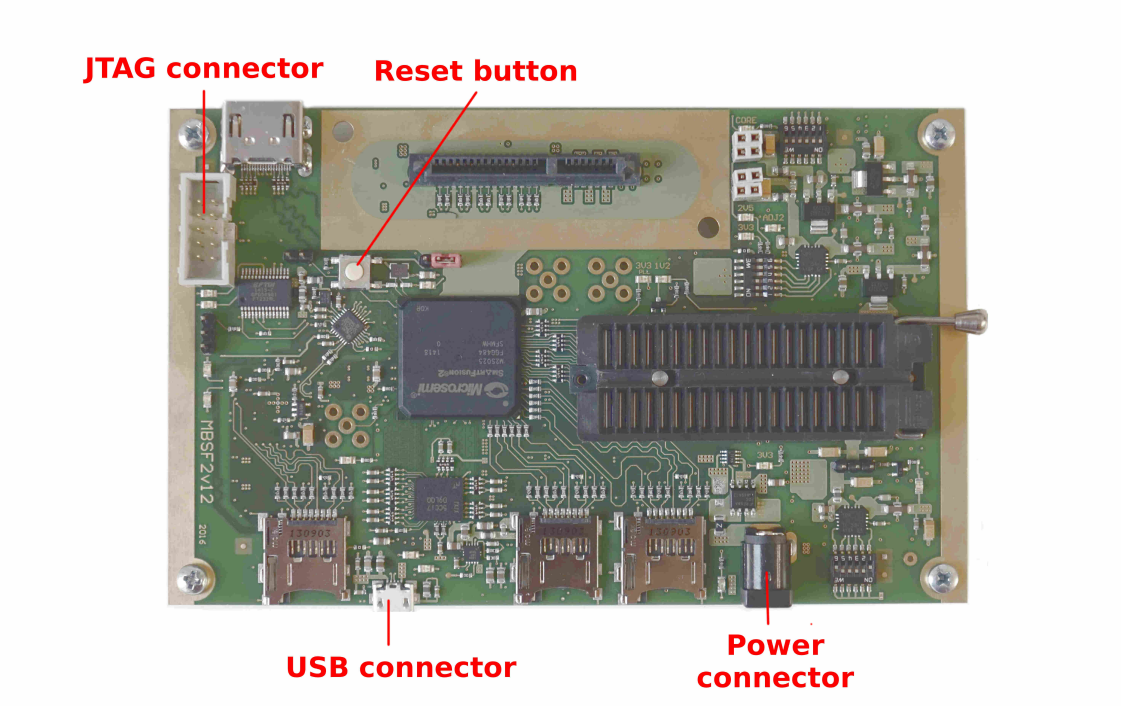


Setting-up power supplies for daughter boards (2/2)

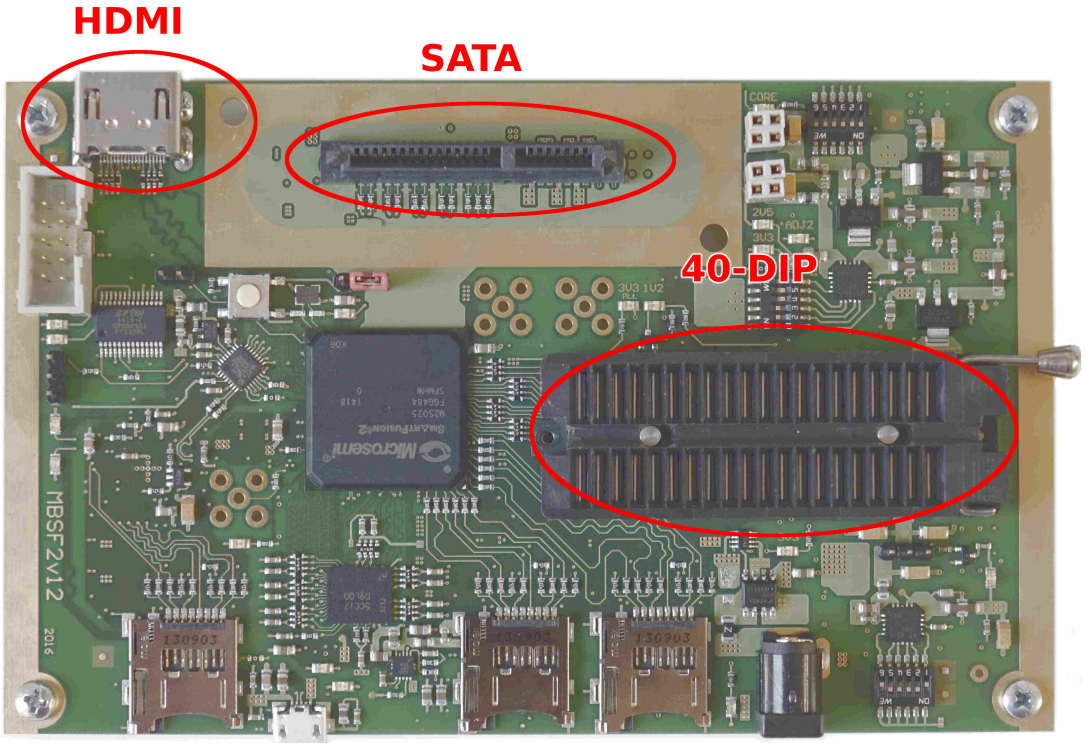
Configuration for daughter boards
SmartFusion 2 & Spartan 6



PC connections and board reset



Daughter board connections



Signal	DATA 0		DATA 1		DATA 2		DATA 3		IO 0	IO 1	IO 2
	P	N	P	N	P	N	P	N			
SATA pin	AB15	AA15	AB13	AB14	AA11	AB11	AA10	AB10	Y9	W10	W9
HDMI pin	Y18	Y19	AB18	AB19	AA17	AB18	Y17	W17	Y20	U18	V17

Outline

PC communicates with the HECTOR motherboard via two channels

- UART – for commands, status, small data (32-bit data blocks)
  - 115 200 baud
  - 8 data bits, 1 start bit, 1 stop bit
  - No parity
- USB mass-storage device – transfer of large files
  - Does not need additional drivers
  - Files can be uploaded to and downloaded from the device
  - Files can be up to 30 MB in size

UART communication protocol

- Packet based – 12-byte packets sent and received
- PC is a master device – it has to initialize every communication
- Communication based on the command-response pairs:
  - Each command generates a response
  - New command cannot be sent before the response to the previous one has been received

Packet types

Command packets:

Control packet – sends commands to the processor

Fabric packet – sends commands and data to the application controller (in the FPGA fabric)

Read file packet – starts the operation and saves its output data to a file

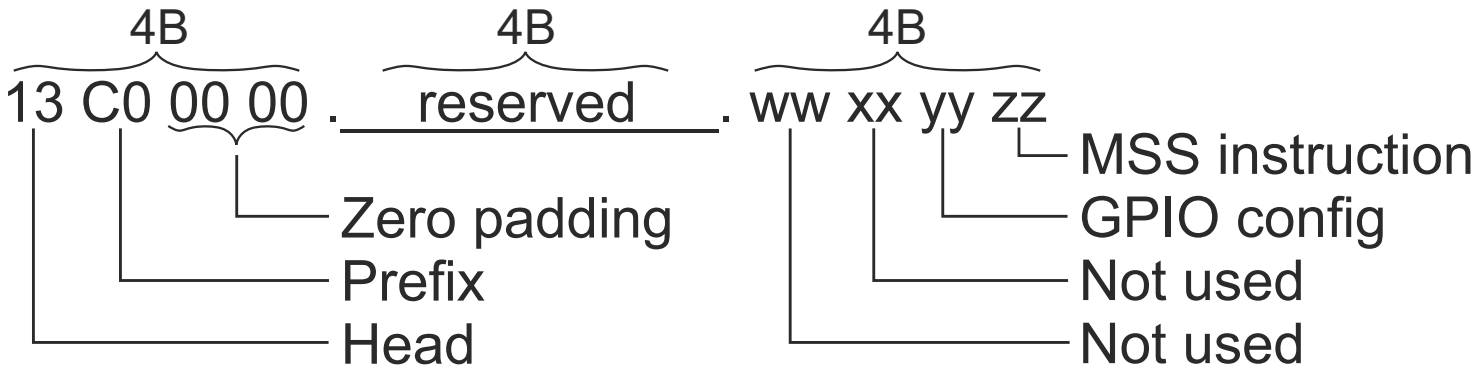
Write file packet – loads the input file into the RAM

Response packets:

Status packet – returns the firmware and acquisition status

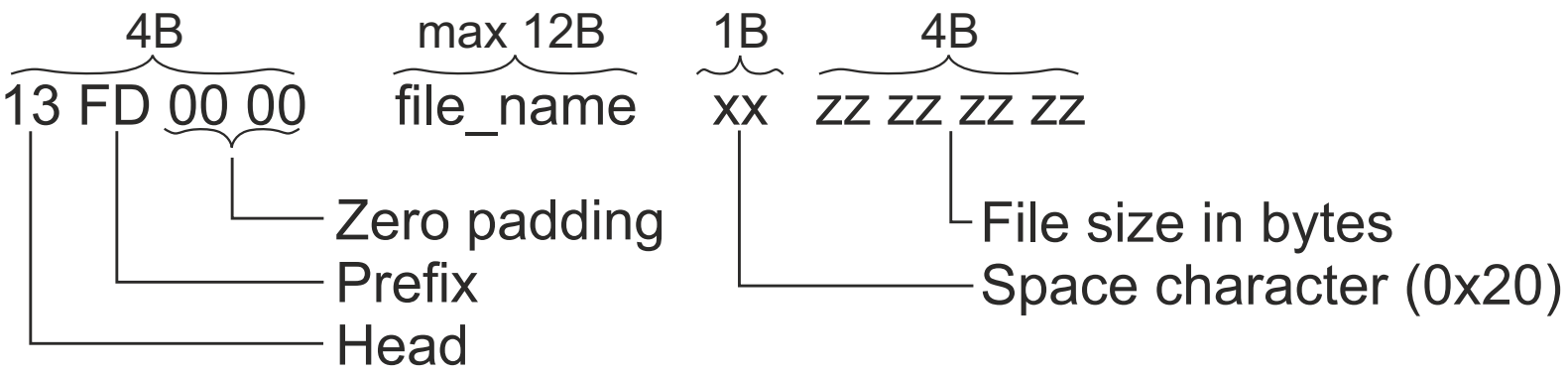
Fabric status packet – returns the status and data from the FPGA fabric controller

Control packet



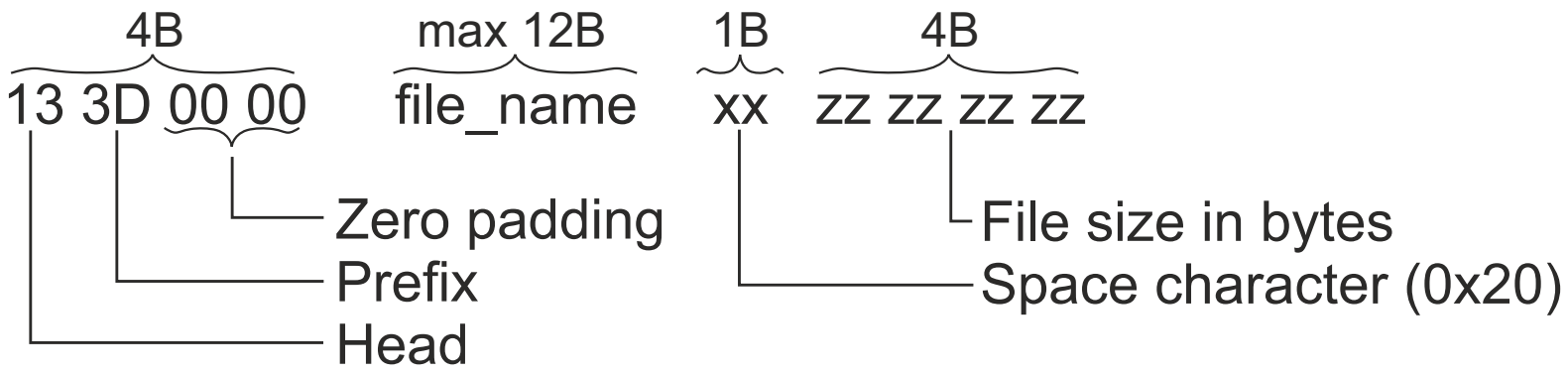
- The MSS instruction tells the firmware what to do
- The GPIO config depends on the instruction:
  - GPIO direction
  - GPIO value
  - Not used if the instruction is not GPIO related
- Response to the Control packet is a Status packet

Read file packet



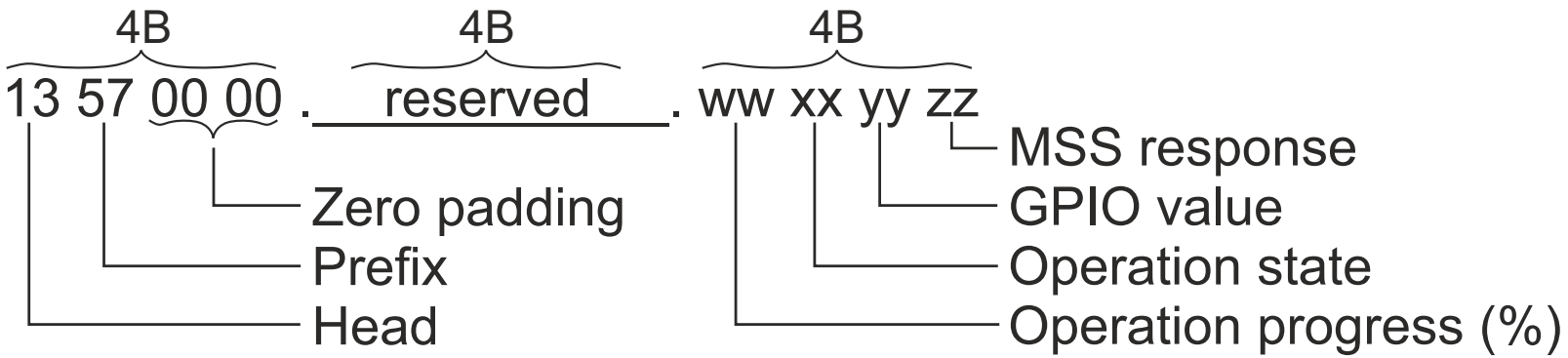
- Starts acquisition of a data file
- The filename is at most 12 characters long
- The specified filename will be used to save the file on the HECTOR disk
- The file size is given in the big endian coding style
- The response to the Read file packet is a Status packet

Write file packet



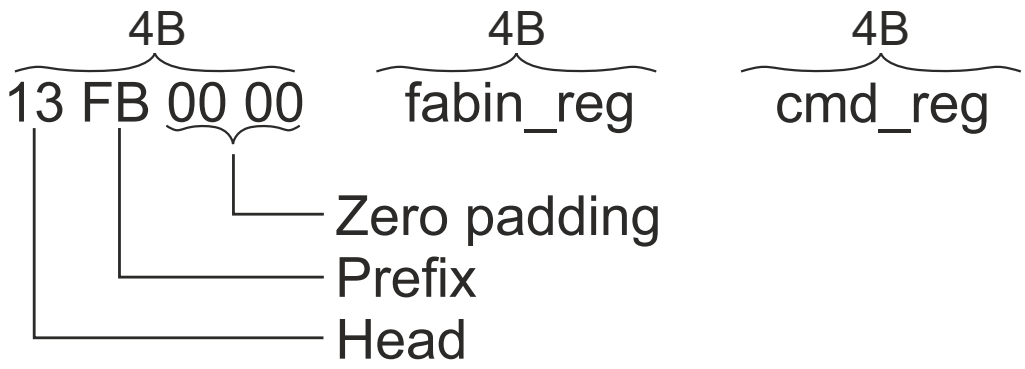
- Loads the file from the HECTOR disk into the RAM
- The filename is at most 12 characters long
- The File size is given in the big endian coding style
- Response to the Write file packet is a Status packet

Status packet



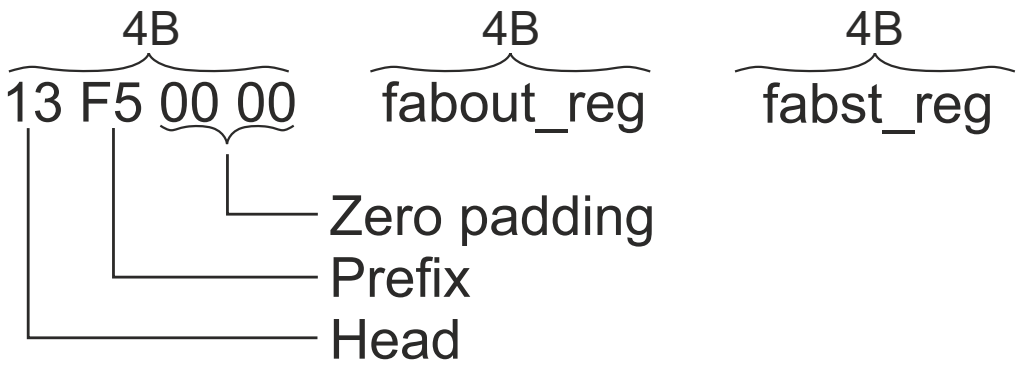
- Status packet is returned after every command

Fabric packet



- Sends a 32-bit command and 32-bit data to the FPGA controller
- Both fields (command and data) are coded in big endian
- Response to the Fabric packet is a Fabric status packet

Fabric status packet



- Returns the status of the FPGA controller and a 32-bit data word from the FPGA fabric
- Both fields are coded in big endian

TCL interface to the HECTOR board

- TCL interpreter is embedded in every EDA tool (including Libero)
- Easy to manipulate strings and files
- We provide TCL language extension for HECTOR evaluation board (in HECTOR_data_acq.tcl)

```
# Include HECTOR TCL extension
source HECTOR_data_acq.tcl
# Open the serial connection
set deviceHandle [openDevice COM1]
# Get the device status
getStatus $deviceHandle
# Close the serial connection
disconnect $deviceHandle
```

Opening and closing the connection

Open the connection to the board at COM1 and save it to the variable ”deviceHandle”

```
set deviceHandle [openDevice COM1]
```

Close the connection

```
disconnect $deviceHandle
```

Getting the board's status

Read status of the board

```
getStatus $deviceHandle
```

Returns a TCL list which includes:

0. Response to the last command
1. Current operation progress
2. State of the acquisition
3. GPIO value (HEX)
4. FPGA controller status (HEX)
5. FPGA controller data (HEX)

Controlling the GPIOs

Configure GPIOs as inputs or outputs

<value> is an 8-bit decimal value, in which every bit set to 0 represent input and 1 represent output.

```
configureGPIO $deviceHandle <value>
```

Set the GPIO output value

<value> is an 8-bit decimal value, in which bits set as inputs (by the previous command) are ignored.

```
setGPIO $deviceHandle <value>
```

Resetting the board components

Reset the FPGA-based control logic

```
sendFabricReset $deviceHandle
```

Reset the daughter board

```
sendDaughterReset $deviceHandle
```

Reset the FPGA-based control logic as well as motherboard firmware

```
softReset $deviceHandle
```

Selecting the daughter board connector

Select active daughter board connector

```
selectDaughterBoard $deviceHandle <value>
```

<value> is either 1 or 2:
1 corresponds to the HDMI connector, whereas 2 corresponds to the SATA connector.

Sending commands and data to FPGA

Send command and/or data to the control logic block in the FPGA fabric

```
sendFabricCommand $deviceHandle <command> <data>
```

<command> is a 32-bit command sent to the FPGA controller.
<data> is a 32-bit data word sent to the FPGA controller.

Filesystem operations 1/4

Create a filesystem in the DDR RAM

```
createFileSystem $deviceHandle
```

Mount the filesystem to the PC

```
mountDisk $deviceHandle
```

Filesystem operations 2/4

Acquire data to a data file

```
beginAcquisition $deviceHandle <filename> <size>
```

<filename> is a name for the file created in the RAM filesystem. It may contain up to 12 characters.
<size> is the size of the file in bytes.

Load a data file from the filesystem to the RAM

```
loadInputFile $deviceHandle <filename> <size>
```

<filename> is the name of the file in the filesystem, which should be loaded to the RAM. It may contain up to 12 characters.
<size> is the size of the file in bytes.

Filesystem operations 3/4

Synchronize disks (finish all read/write operations) to ensure all files are written and intact

```
syncDrives
```

Filesystem operations 4/4

Find the HECTOR drive (its letter, mount point)

```
findHECTOR <timeout>
```

<timeout> is the timeout in seconds. If the timeout passes and the HECTOR disk is not found, the function returns 0 and you have to find the drive (mount it) manually.

Returns a TCL list:

0. Disk letter (device node in Linux)
1. UUID (48A1-0000)
2. Mountpoint

Operating system support

Functions of synchronizing the disks and automatic disk discovery are supported by:

Windows – full support

Linux – need to have access to the `blkid` utility and permissions to sync disks (mount)

MAC – not yet supported

Outline

ARM firmware

Consists of:

- Initialization part
- UART routines
- USB Mass storage routines
- Predefined functions of the controller

Initialization part of the firmware

At start up:

- Configures:
  - GPIO
  - UART
  - RAM memory
  - Timers and interrupts
- Resets FPGA fabric

UART routines in the firmware

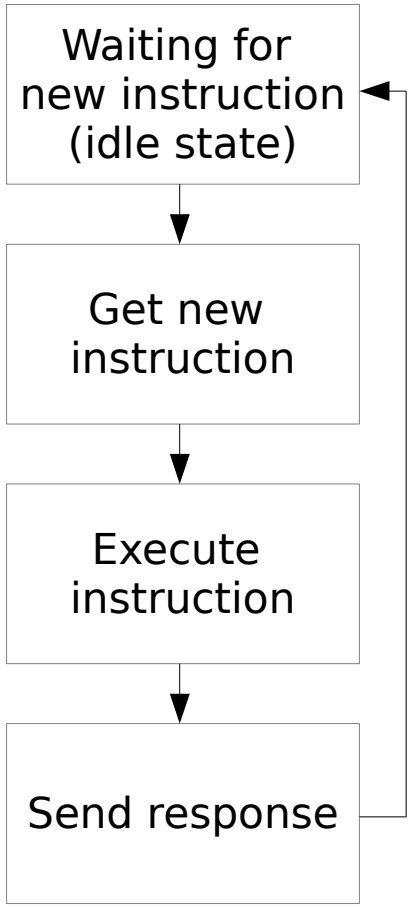
- Recognize the header of the packet
- Collect packet data
- Load new command to the MSS or FPGA
- Load other data words according to the packet type

USB mass storage routines in the firmware

- Use Microsemi drivers
- Based on Microsemi example
- Allow direct access to one half of the external RAM
- Work independently of the main controller routines

Controller with the predefined functions

- Executes received instructions
- Manages operations in FPGA
- Provides simple way of how to insert new instructions
- Consists of several predefined functions called by packets



Predefined functions of the controller (1/5)

MB_IDLE (0x00)

- Default state, no instruction is called
- Controller returns to this state after every instruction
- Check firmware operation state

MB_FAB_CMD (0x01)

- Take a reserved word from the Control packet and sent it to the FPGA
- Obsolete instruction

MB_GET_OP_STATE (0x07)

- Return the firmware status packet
- Useful to see the actual process of the operation

Predefined functions of the controller (2/5)

MAKE_FAB_RST (0x04)

- Assert reset of the FPGA fabric during 1 ms
- State of the operation can not be in the ‘busy’ state
- The reset signal sent to MSS GPIO 8

MAKE_GEN_RST (0x05)

- Assert reset of the generator (or other entity) during 1 ms
- The reset signal sent to MSS GPIO 9

MB_OP_RST (0x0B)

- Resets the state of the internal operation and the FPGA fabric (same as MAKE_FAB_RST instruction)

Predefined functions of the controller (3/5)

GPIO 0 - 7 are configured by the following instructions:

MB_GPIO_CFG (0x09)

- Set appropriate direction of GPIO according to the GPIO config byte from the Control packet
- 0 - input, 1 - output (GPIO 0 is the LSB bit)

MB_GPIO_SET (0x0A)

- Set appropriate value of GPIO according to the GPIO config byte from the Control packet
- 0 - low, 1 - high (GPIO 0 is the LSB bit)

Predefined functions of the controller (4/5)

MB_MAKE_FSYS (0x08)

- Create the file system
- Can be used to delete every file from the HECTOR disk

MB_MOUNT_DISK (0x0A)

- Configure the Mass storage interface and connect the HECTOR disk to the Host PC
- Command can be used to refresh the contents of the HECTOR disk seen by the Host PC

Predefined functions of the controller (5/5)

GEN_DATA_COLLECT (0x02)

- Function called by the Read file packet
- Write data to the WSIZE and WSTART FPGA registers and start execution of the FPGA command

MB_LOAD_FILE (0x0D)

- Function called by the Write file packet
- Write data to the RSIZE and RSTART FPGA registers and copy data from the file system to RAM

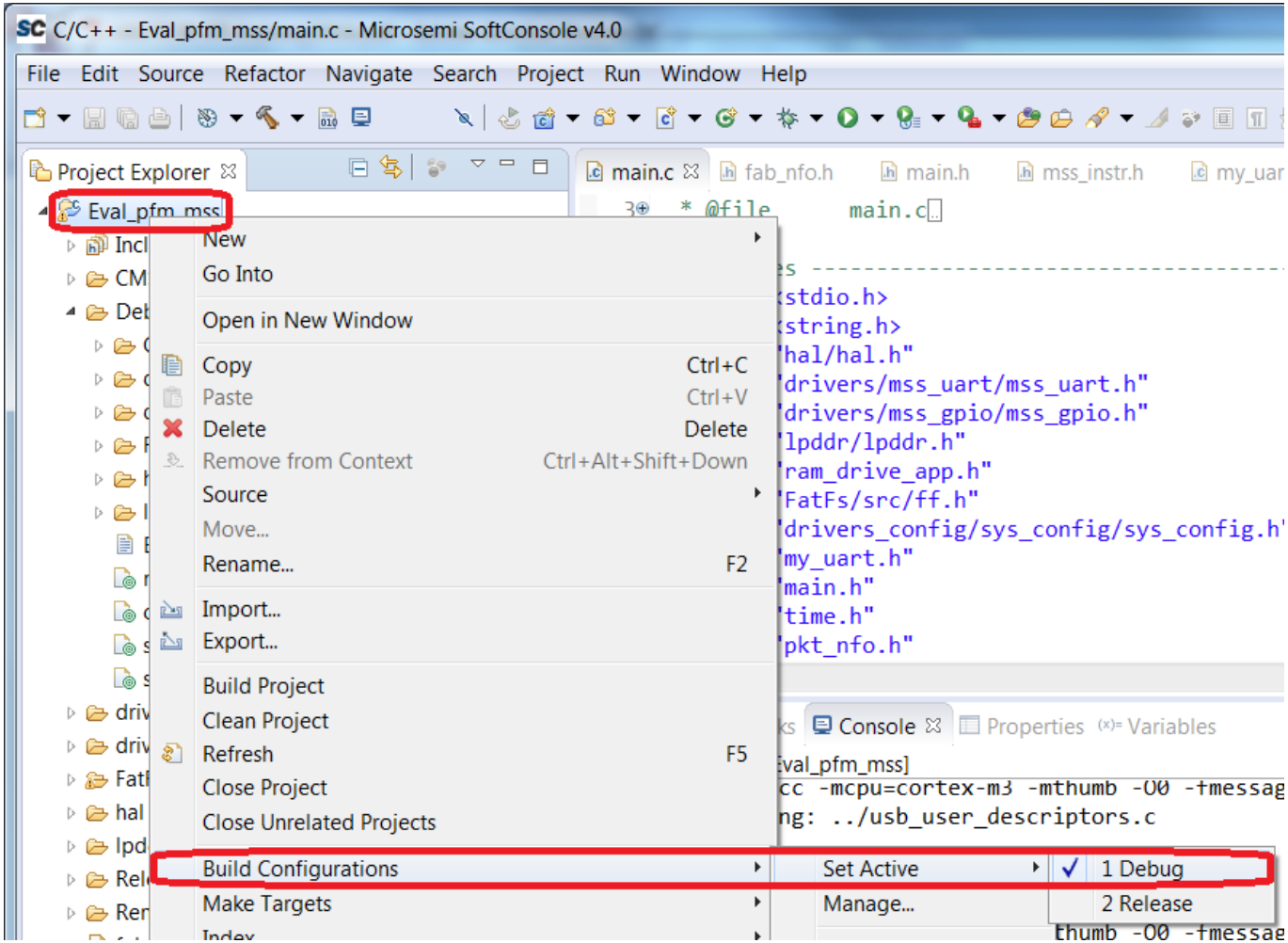
SEND_FAB_RSP (0x0C)

- Function called by the Fabric packet
- Writes data to the FABIN and FPGA CMD registers and returns values of the FABOUT and FABST registers

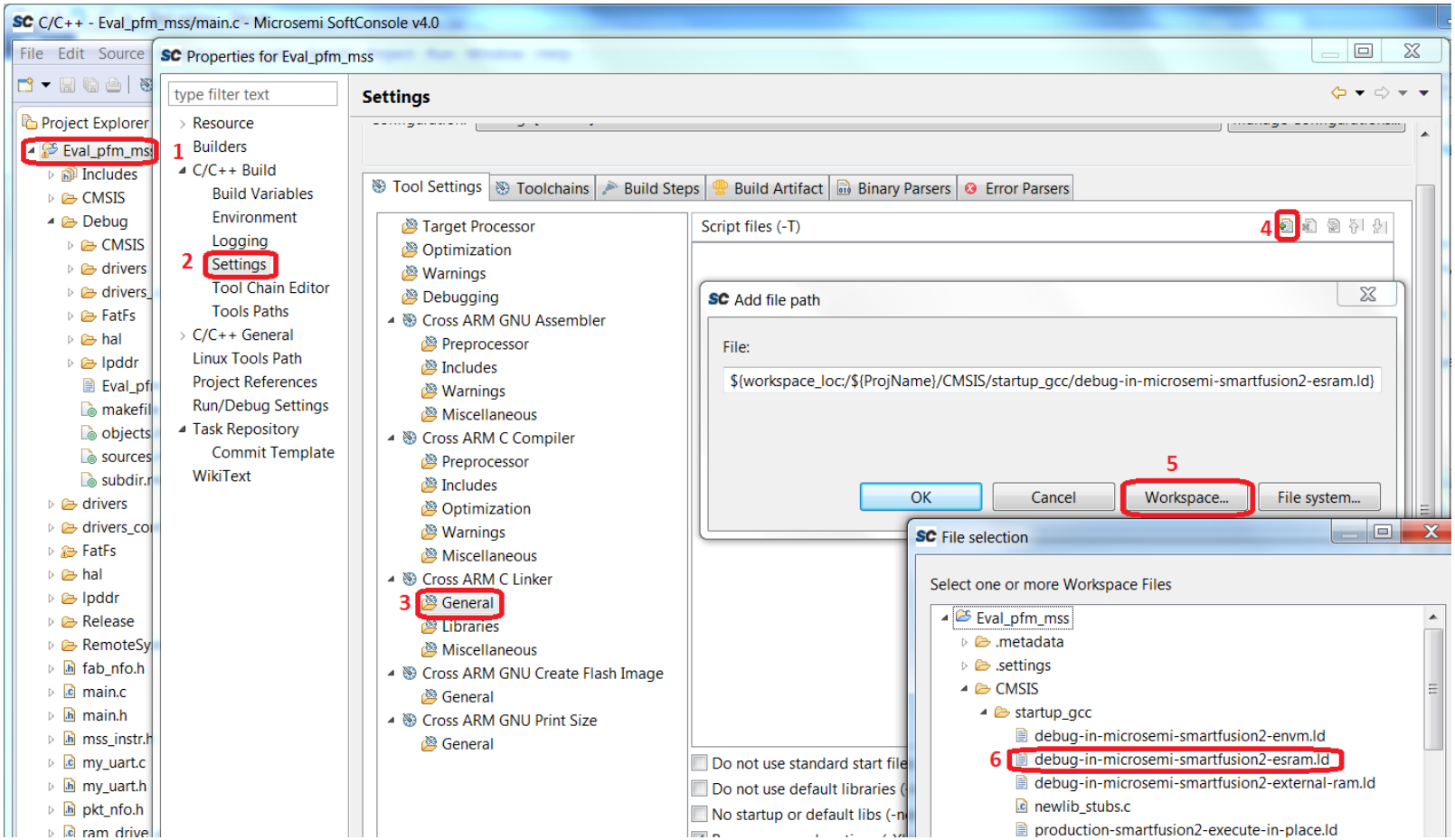
Firmware debugging

- Build a binary for debugging
- Create the Debug Launch Configuration
- Run Debug

Activate Debug Build Configuration

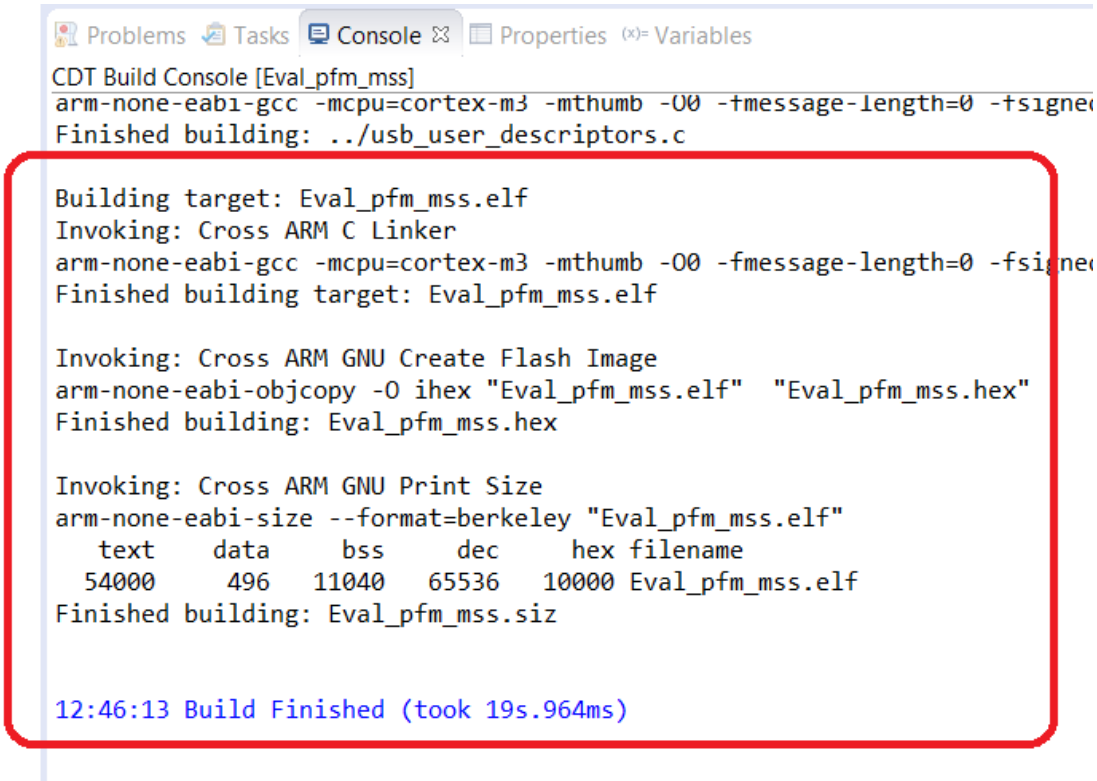


Add/Modify Linker script



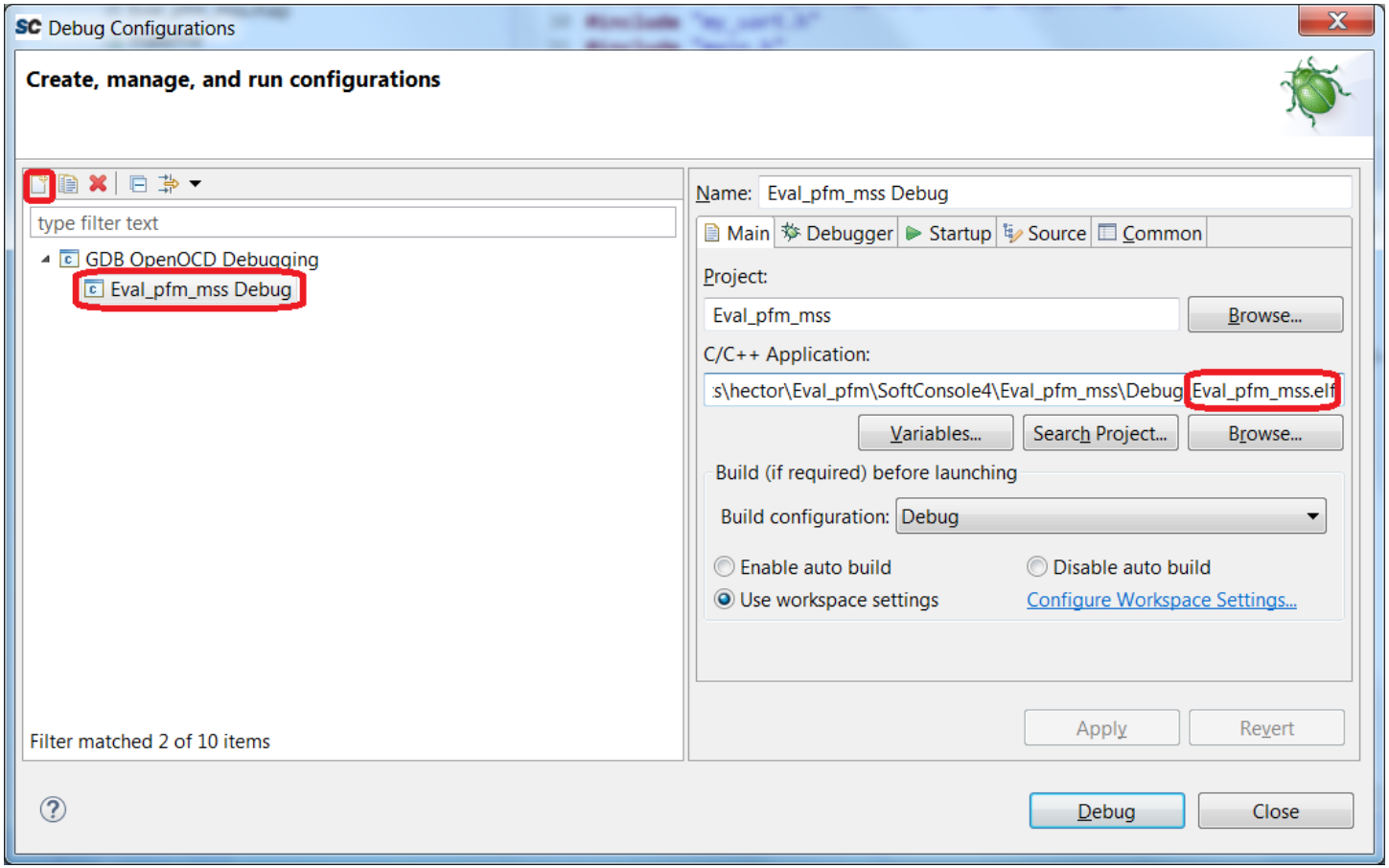
Build a binary for debugging

- (Clean Project)
- <Ctrl>–<B>

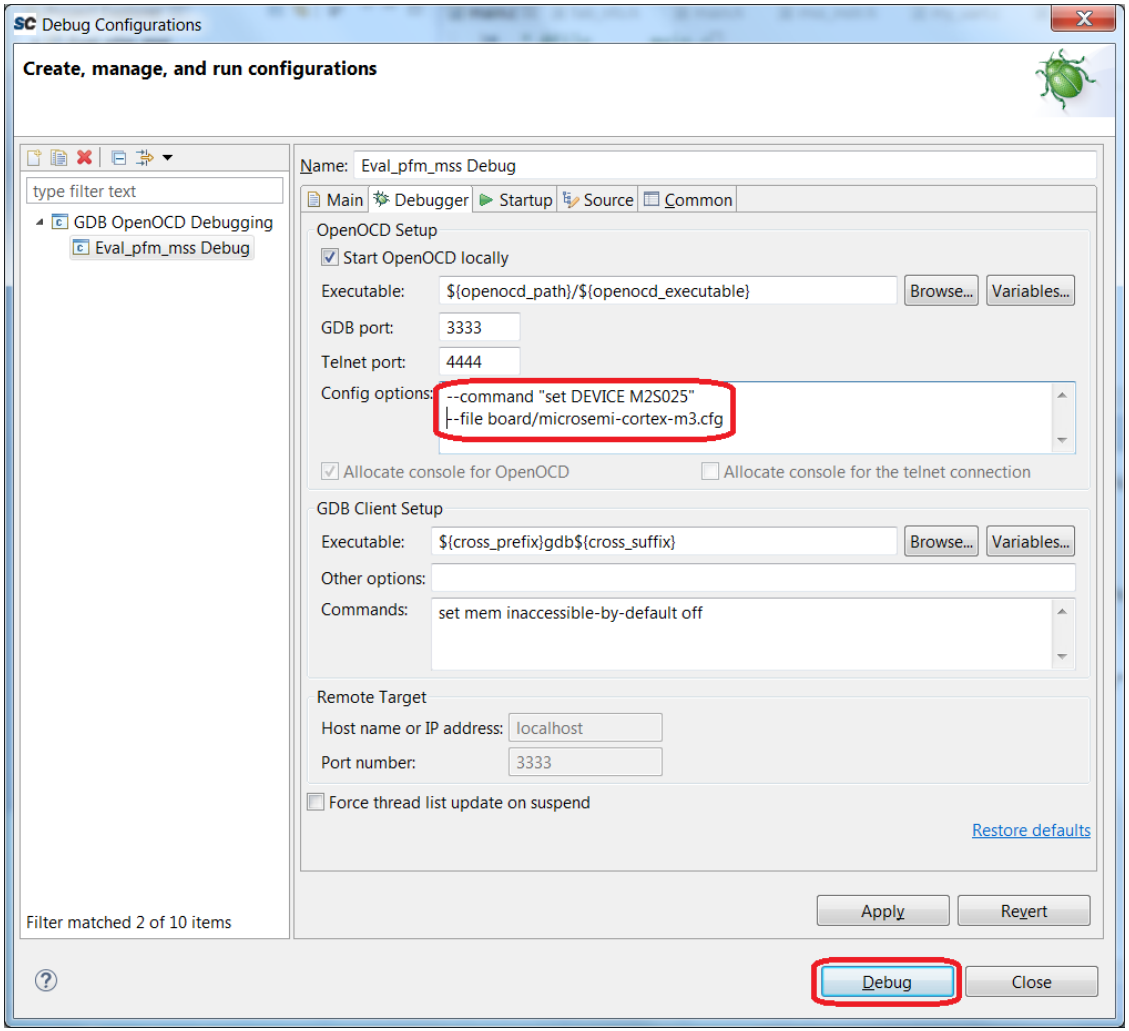


Create a debug configuration

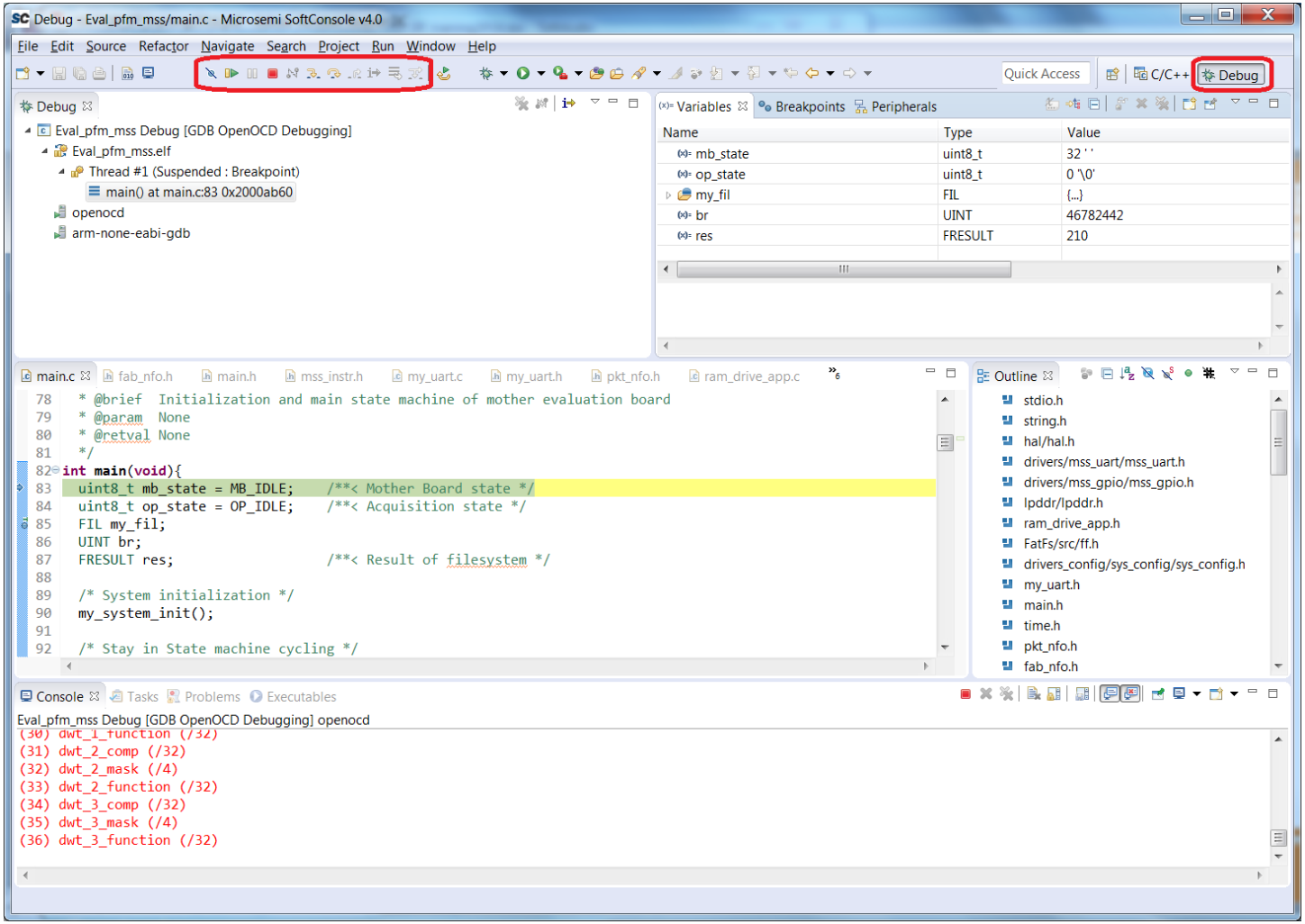
- <Run>–<Debug Configurations>



Setup and run the debug configuration



Debugging


Debug - Eval_pfm_mss/main.c - Microsemi SoftConsole v4.0
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
C/C++
Debug
Debug
Eval_pfm_mss Debug [GDB OpenOCD Debugging]
Eval_pfm_mss.elf
Thread #1 (Suspended : Breakpoint)
main() at main.c:83 0x2000ab60
openocd
arm-none-eabi-gdb
Variables
Breakpoints
Peripherals
Name
Type
Value
mb_state
uint8_t
32 ' '
op_state
uint8_t
0 '\0'
my_fil
FIL
{...}
br
UINT
46782442
res
FRESULT
210
main.c
fab_nfo.h
main.h
mss_instr.h
my_uart.c
my_uart.h
pkt_nfo.h
ram_drive_app.c
78 * @brief Initialization and main state machine of mother evaluation board
79 * @param None
80 * @retval None
81 */
82 int main(void){
83 uint8_t mb_state = MB_IDLE; /**< Mother Board state */
84 uint8_t op_state = OP_IDLE; /**< Acquisition state */
85 FIL my_fil;
86 UINT br;
87 FRESULT res; /**< Result of filesystem */
88
89 /* System initialization */
90 my_system_init();
91
92 /* Stay in State machine cycling */
Outline
stdio.h
string.h
hal/hal.h
drivers/mss_uart/mss_uart.h
drivers/mss_gpio/mss_gpio.h
lpddr/lpddr.h
ram_drive_app.h
FatFs/src/ff.h
drivers_config/sys_config/sys_config.h
my_uart.h
main.h
time.h
pkt_nfo.h
fab_nfo.h
Console
Tasks
Problems
Executables
Eval_pfm_mss Debug [GDB OpenOCD Debugging] openocd
(30) dwt_1_function (/32)
(31) dwt_2_comp (/32)
(32) dwt_2_mask (/4)
(33) dwt_2_function (/32)
(34) dwt_3_comp (/32)
(35) dwt_3_mask (/4)
(36) dwt_3_function (/32)

Debugging

- F5 – Step into
- F6 – Step over
- F7 – Step return
- F8 – Resume

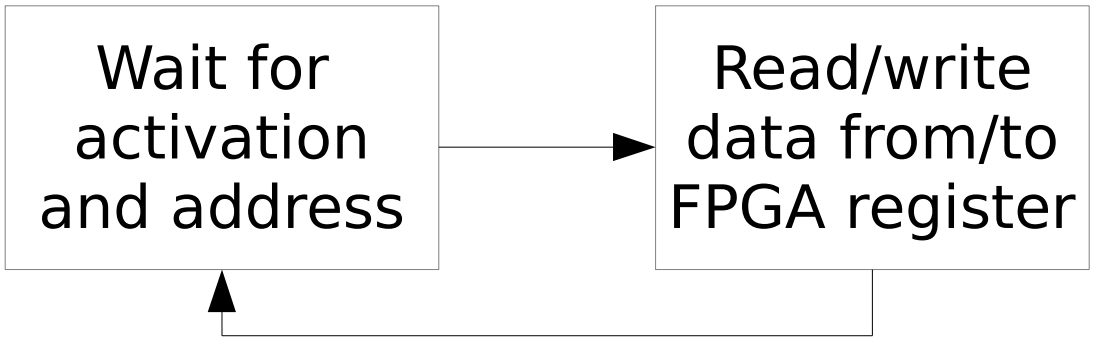
Outline

FPGA fabric controller

- Provides interface between ARM and user entity
- Consists of two components
  - RAM interface
  - FPGA fabric registers
- Two options to control user entity:
  - Command execution state machine
  - User application

FPGA fabric registers (1/4)

- Developed for commands and small data blocks
- Accessible only by ARM and dedicated packets
- For ARM, registers are accessible as an AHB slave device
- Controlled by the *ahb_reg.vhd* component in the control unit



FPGA fabric registers (2/4)

cmd_reg

- Command register accessible by the Fabric packet
- If a new value is written to the cmd_reg, then the new_cmd flag is asserted
- Allows to execute various operations in the FPGA fabric
- Directly accessible to the Fabric packet (the second 32-bit word of the packet is written to the register)

state_reg

- State register providing information about the FPGA for the MSS and subsequently for the host PC

stage_reg

- Stage register showing actual read/write address of RAM

FPGA fabric registers (3/4)

fabin_reg

- FPGA fabric input register (dir: MSS $\rightarrow$ FPGA fabric)
- Directly accessible to the Fabric packet (the first 32-bit word of the packet is written to the register)

fabout_reg

- FPGA fabric output register (dir: FPGA fabric $\rightarrow$ MSS)
- The value sent in the Fabric status packet (the first 32-bit word of the packet)

fabst_reg

- Used to monitor internal FPGA state
- The value sent in the Fabric status packet (the second 32-bit word of the packet)

FPGA fabric registers (4/4)

wstart_reg, wsize_reg

- Parameters defining position and size of the user file (data in RAM)
- wstart_reg is the start address of the data stream and wsize_reg is the size of the stream

rstart_reg, rsize_reg

- rstart_reg is the start address of the stream and rsize_reg is the size of the stream

user1_reg, user2_reg

- Reserved registers, not used, yet

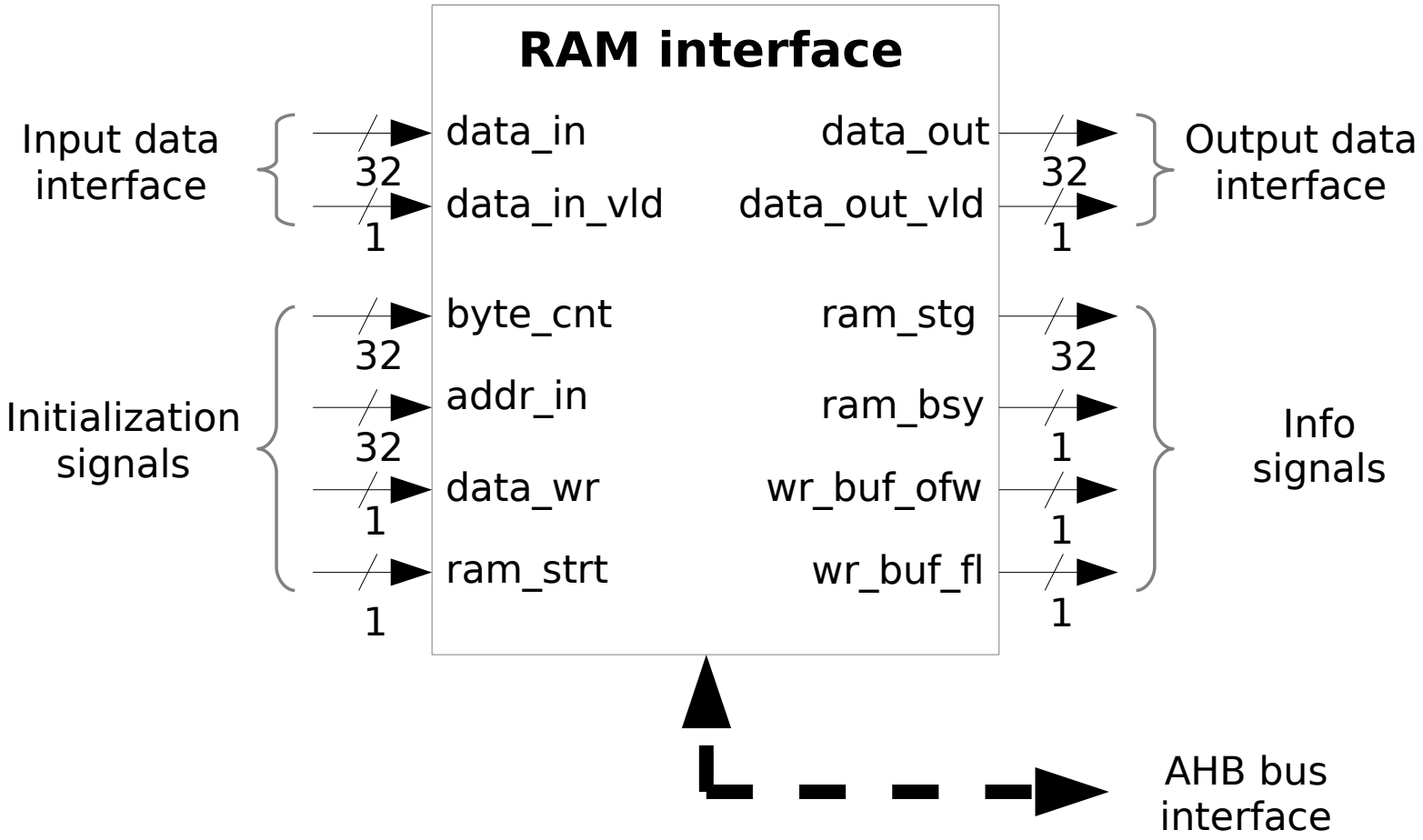
RAM interface

- DMA based transfer for large data depending on the start address and number of required bytes
- Data are written to the external RAM via the AHB bus, independently from ARM
- Controlled by the *ram_if.vhd* component in the control unit

Initialization signals:

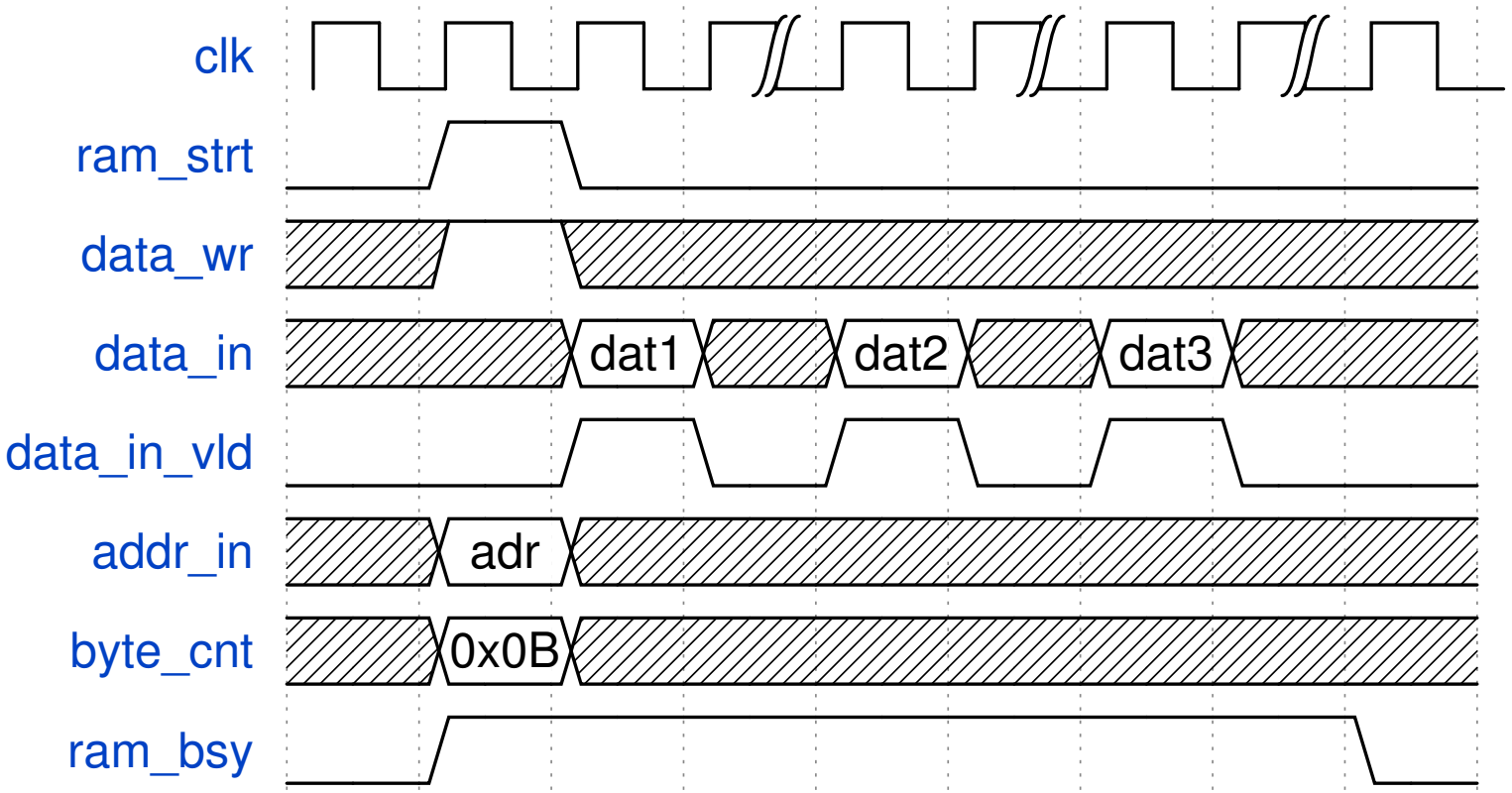
- **byte_cnt** - Number of bytes to read/write
- **addr_in** - Start address
- **data_wr** - Type of operation (read/write)
- **ram_strt** - Starting of the RAM operation

RAM interface



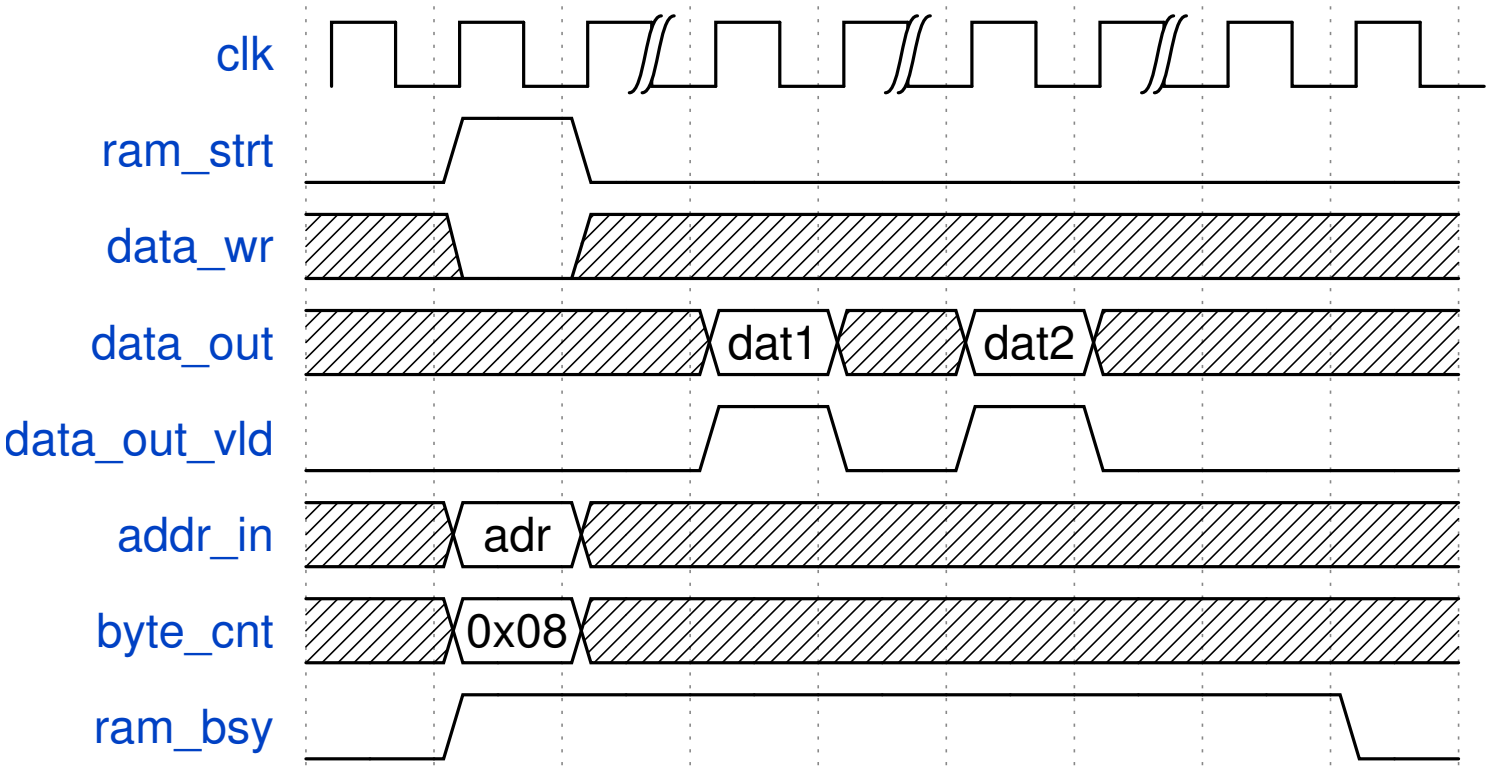
RAM interface - Data writing

- **data_in** - Input data
- **data_in_vld** - Input data validation signal



RAM interface - Data reading

- **data_out** - Output data
- **data_out_vld** - Output data validation signal



RAM interface - Information signals

- **ram_stg** - RAM stage (currently used address)
- **ram_bsy** - Ram busy flag
- **wr_buf_ofw** - Write fifo overflowed
- **wr_buf_fl** - Write fifo full

Writing to the RAM interface - data are inserted to the FIFO

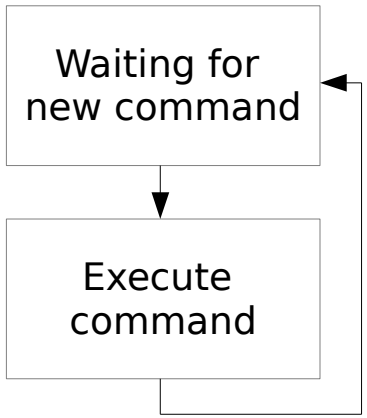
Reading from the RAM interface - data are present directly at the output of the block no FIFO is used

Command execution state machine

- Controlled directly by the Fabric packet
- Simple state machine controlling various operations
- Commands depend on the user application

Fixed commands:

  - Idle command (0x00) - do nothing
  - ‘Operation start’ command (0x01) - used by ARM to run an operation



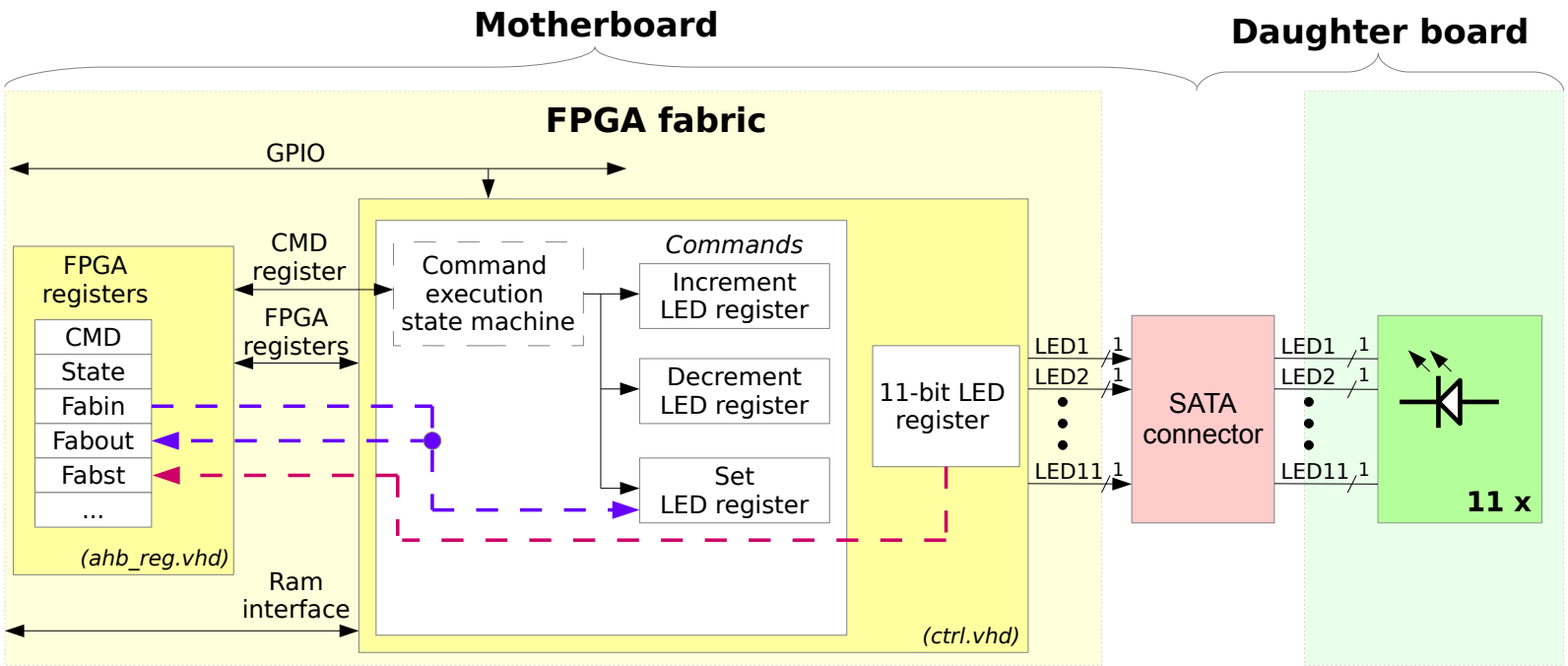
Outline

Outline

Reference design 1

Objective

Control output bits of the motherboard from the script and switch LEDs on and off



Blinking LEDs

- Uses the daughter board with LEDs
- The Fabric packet is used to let LEDs blink
- LEDs are connected to an 11-bit register
- Command execution state machine allows to:
  - Set the LED register (0x06)
  - Increment the LED register (0x04)
  - Decrement the LED register (0x05)
- The Fabin register is connected to the Fabout register
- The Fabst register is the LED register

Workflow

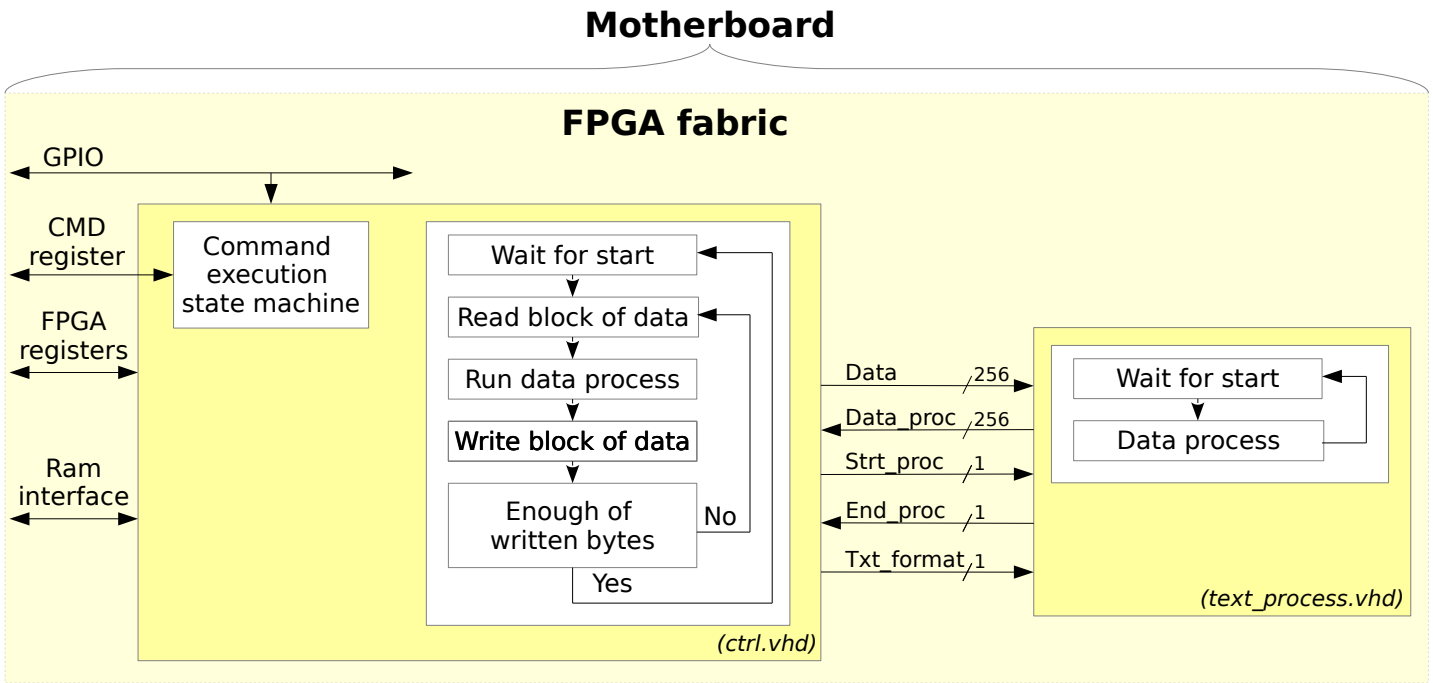
1. Open the project
2. Accept updates (if any)
3. Compile the project (if needed)
4. Program FPGA fabric
5. Run the script

Outline

Reference design 2

Objective

Send data to the processing block → process data → read processed data



Necessary script steps

1. Send the Control packet to make the file system
2. Send the Control packet to mount the HECTOR disk
3. Copy text file to the HECTOR disk
4. Unmount or synchronize the HECTOR disk
5. Send the Write file packet to copy data from file to RAM
6. Send the Fabric packet to set mode of the text processing
7. Send the Read file packet to run operation and set size and name of required file
8. Repeatedly send the Control packet to get the state of the operation
9. Send the Control packet to mount the HECTOR disk
10. Copy the file from the HECTOR disk

Workflow

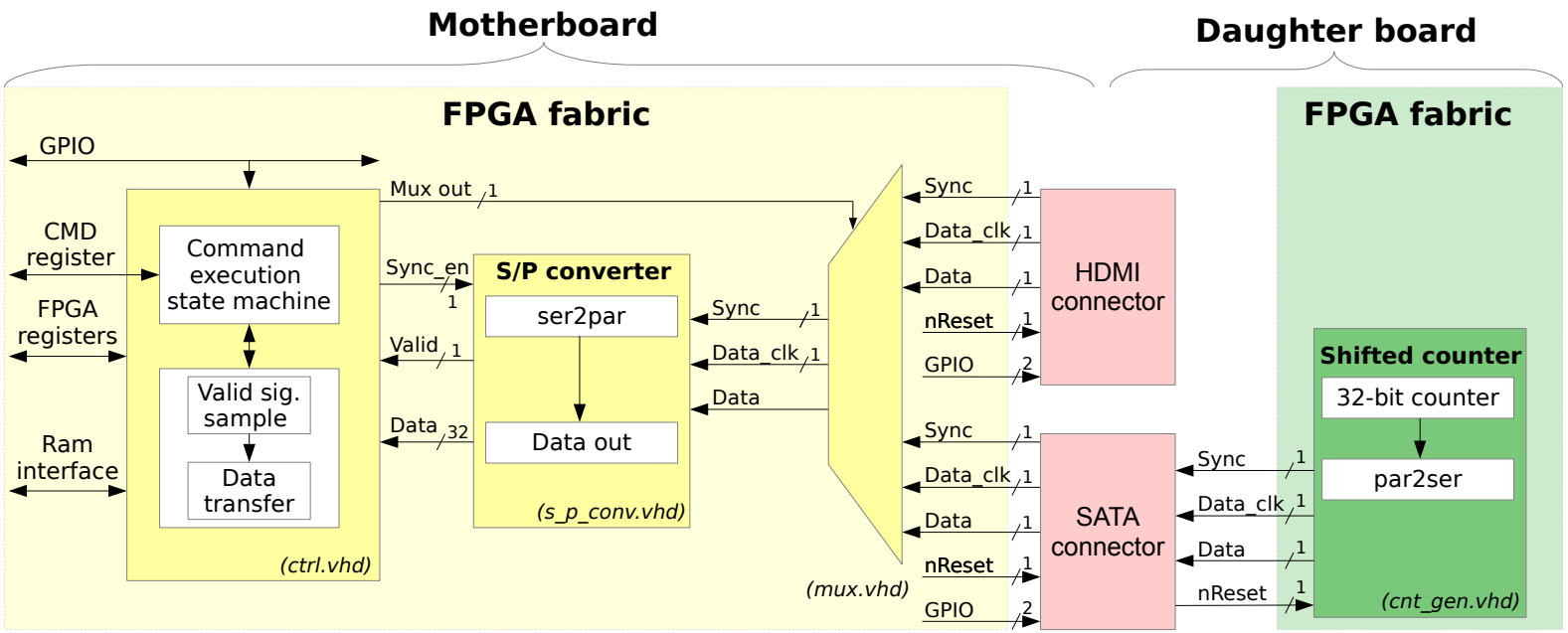
1. Open the project
2. Accept updates (if any)
3. Compile the project (if needed)
4. Program FPGA fabric
5. Run the script

Outline

Reference design 3

Objective

Implement a simple 32-bit counter in the daughter board and read the counter values into the file



Necessary script steps

1. Send the Control packet to make the file system
2. Send the Fabric packet to choose the connector
3. Send the Fabric packet to enable a synchronization
4. Send the Read file packet to run operation and set size and name of the required file
5. Repeatedly send the Control packet to get the state of the operation
6. Send the Control packet to mount the HECTOR disk
7. Copy the file from the HECTOR disk

Workflow

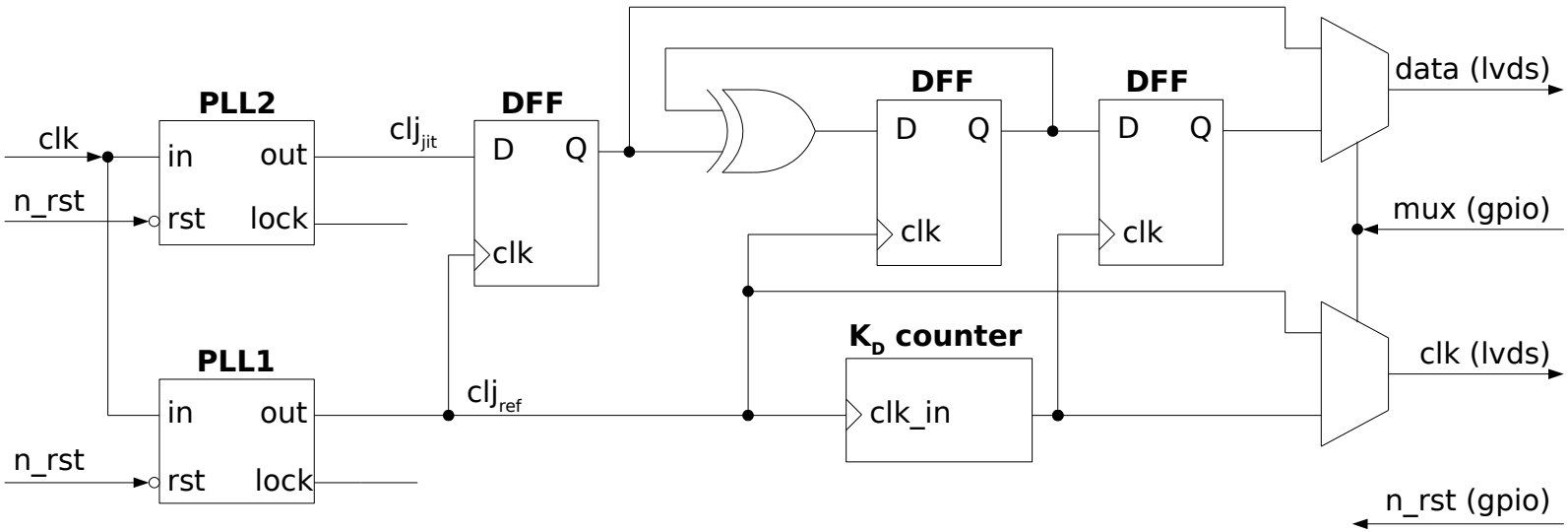
1. Open the project
2. Accept updates (if any)
3. Compile the project (if needed)
4. Program FPGA fabric
5. Run the script

Outline

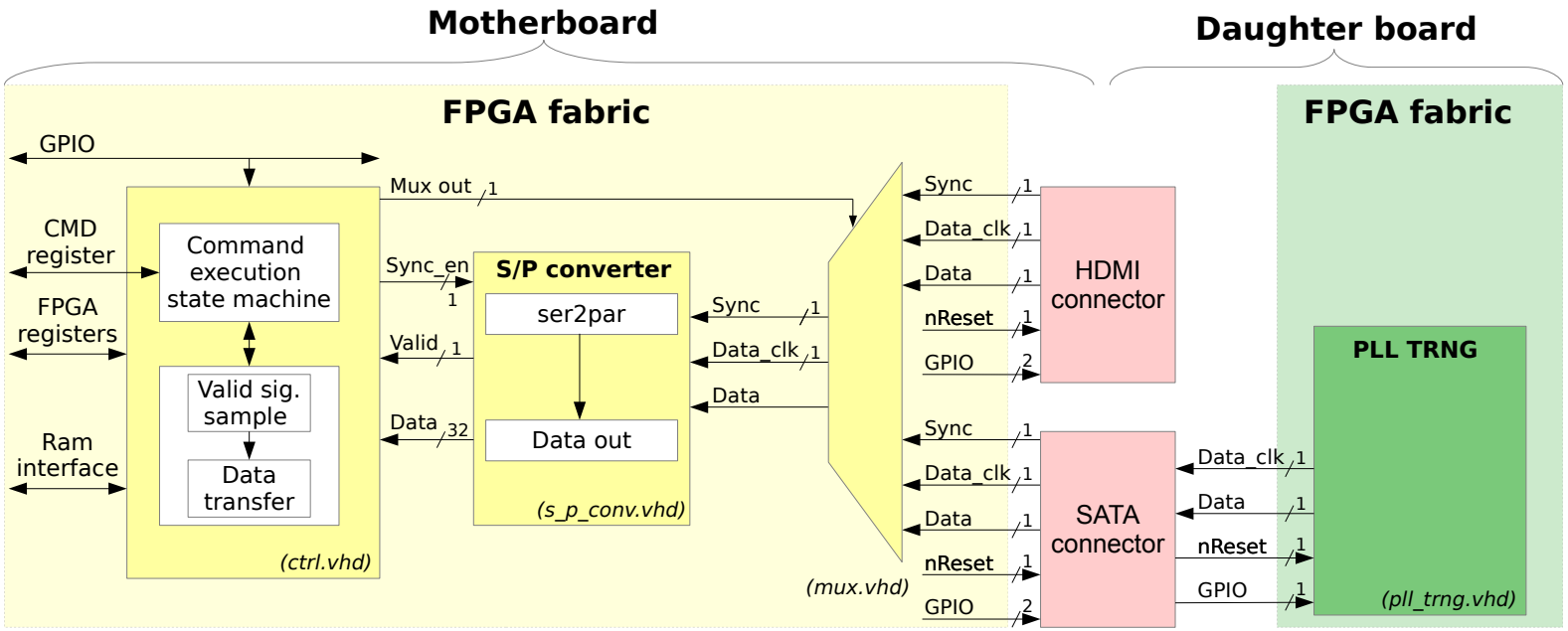
Reference Design 4

Objective

Implement simple RNG (PLL-TRNG) and read random data before and after the decimator, using a multiplexer controlled by the script (mux(gpio))



Hardware structure of the project



Workflow

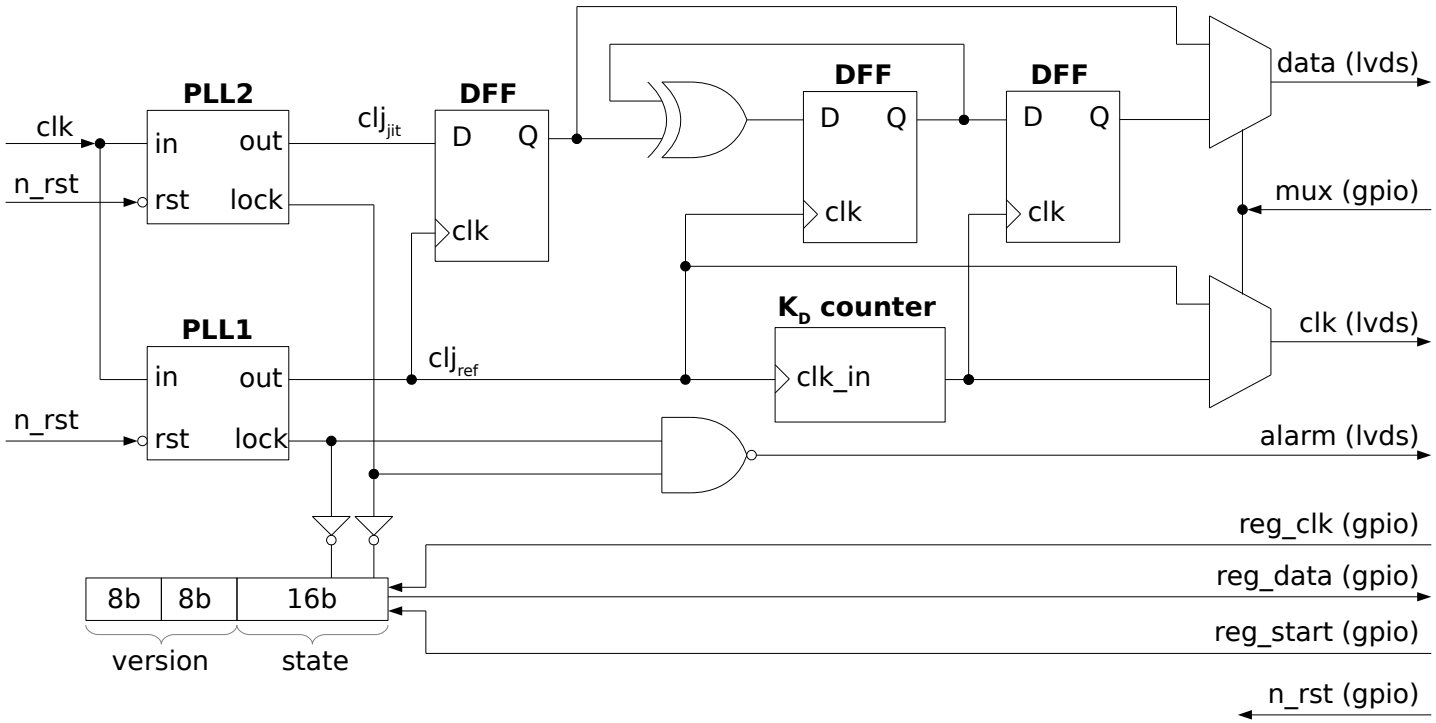
1. Open the project
2. Accept updates (if any)
3. Compile the project (if needed)
4. Program FPGA fabric
5. Run the script

Outline

Let you make your own design

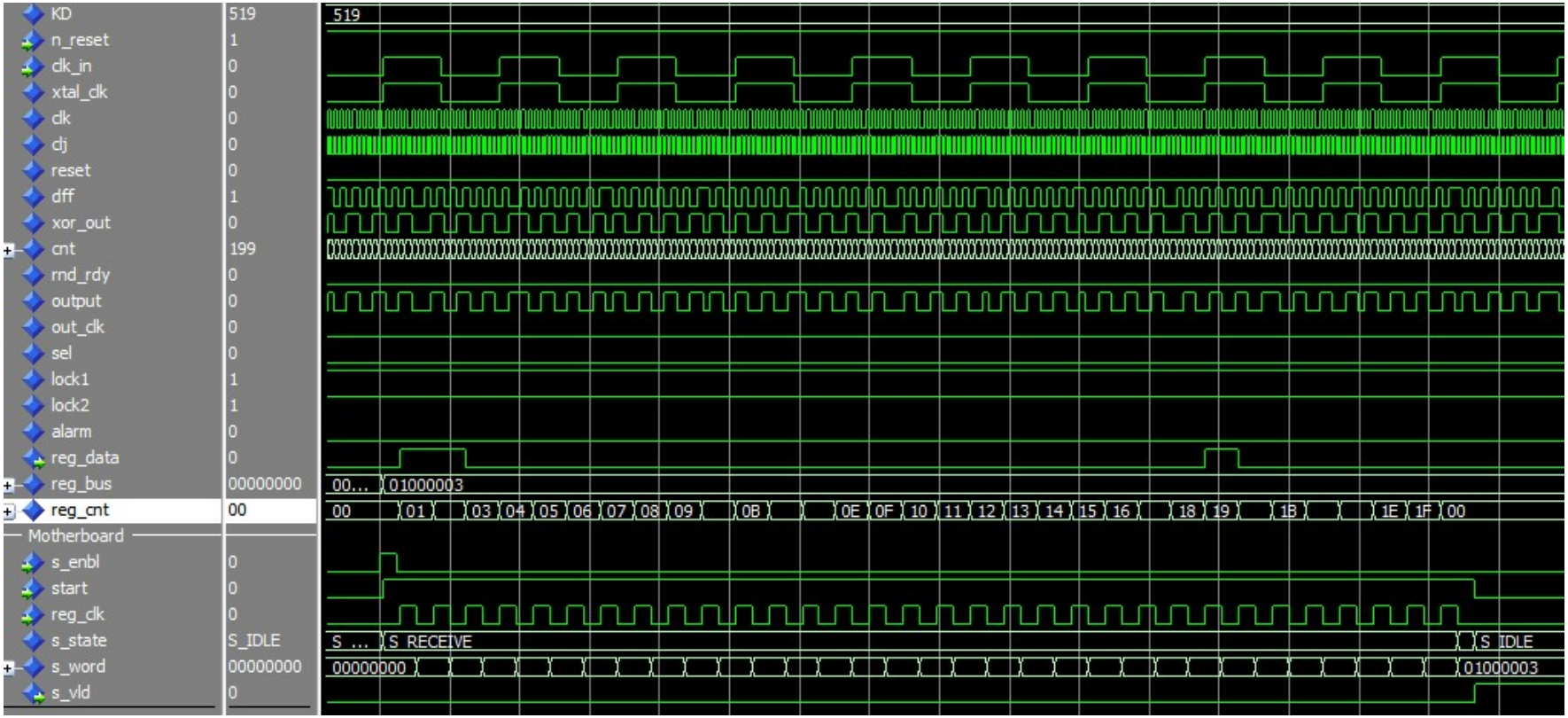
Objective

Implement the PLL-TRNG including simple tests with alarm and state word outputs

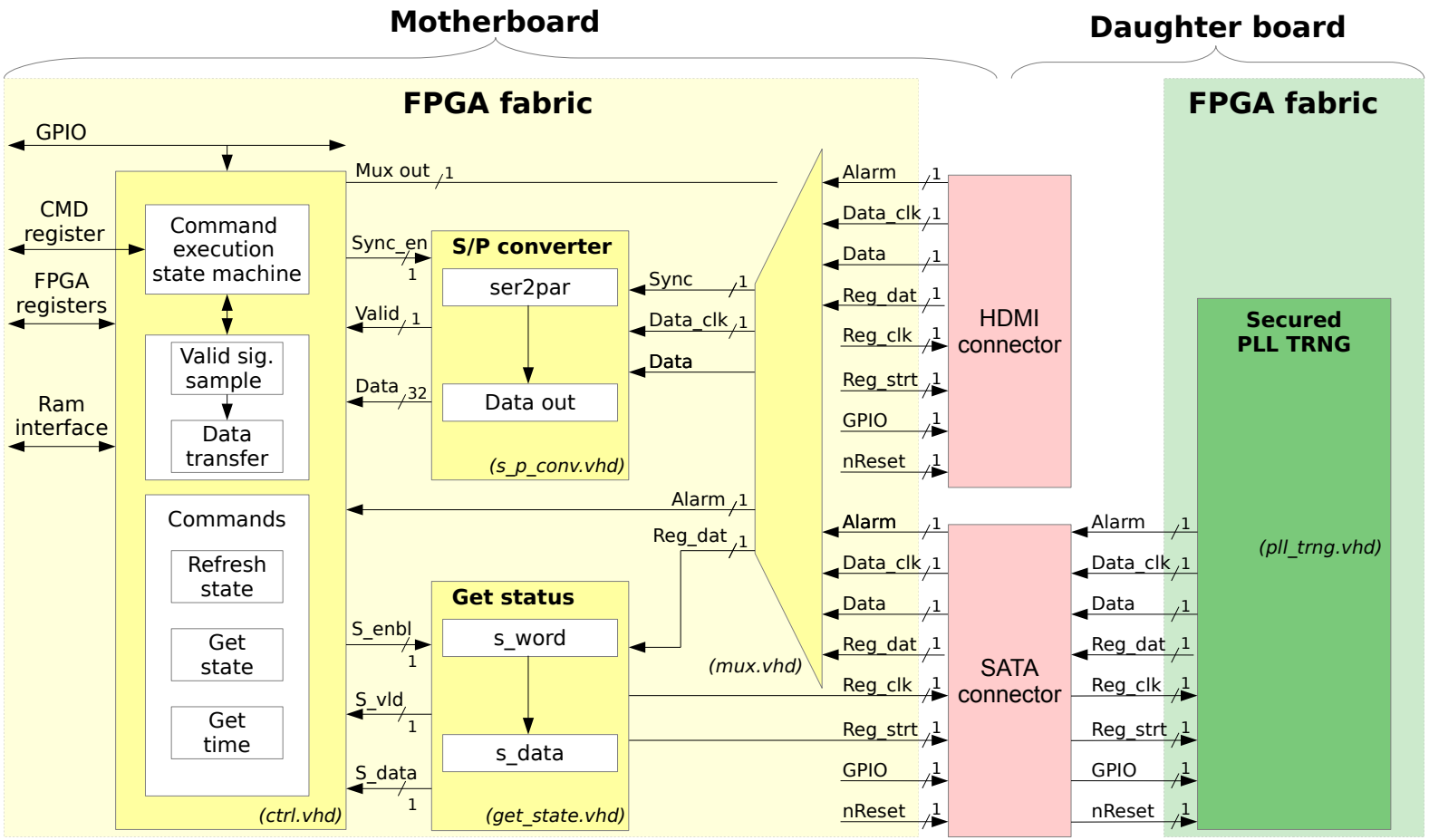


Communication waveforms

- Script launches the request to read the state (*s_enbl*)
- Motherboard starts the communication and generates clock (*start, reg_clk*)
- Daughter board samples the state and sends out 32 bits (*reg_data*)
- Receiver confirms reception of the state word to the firmware (*s_vld*)



Hardware structure of the project



Outline

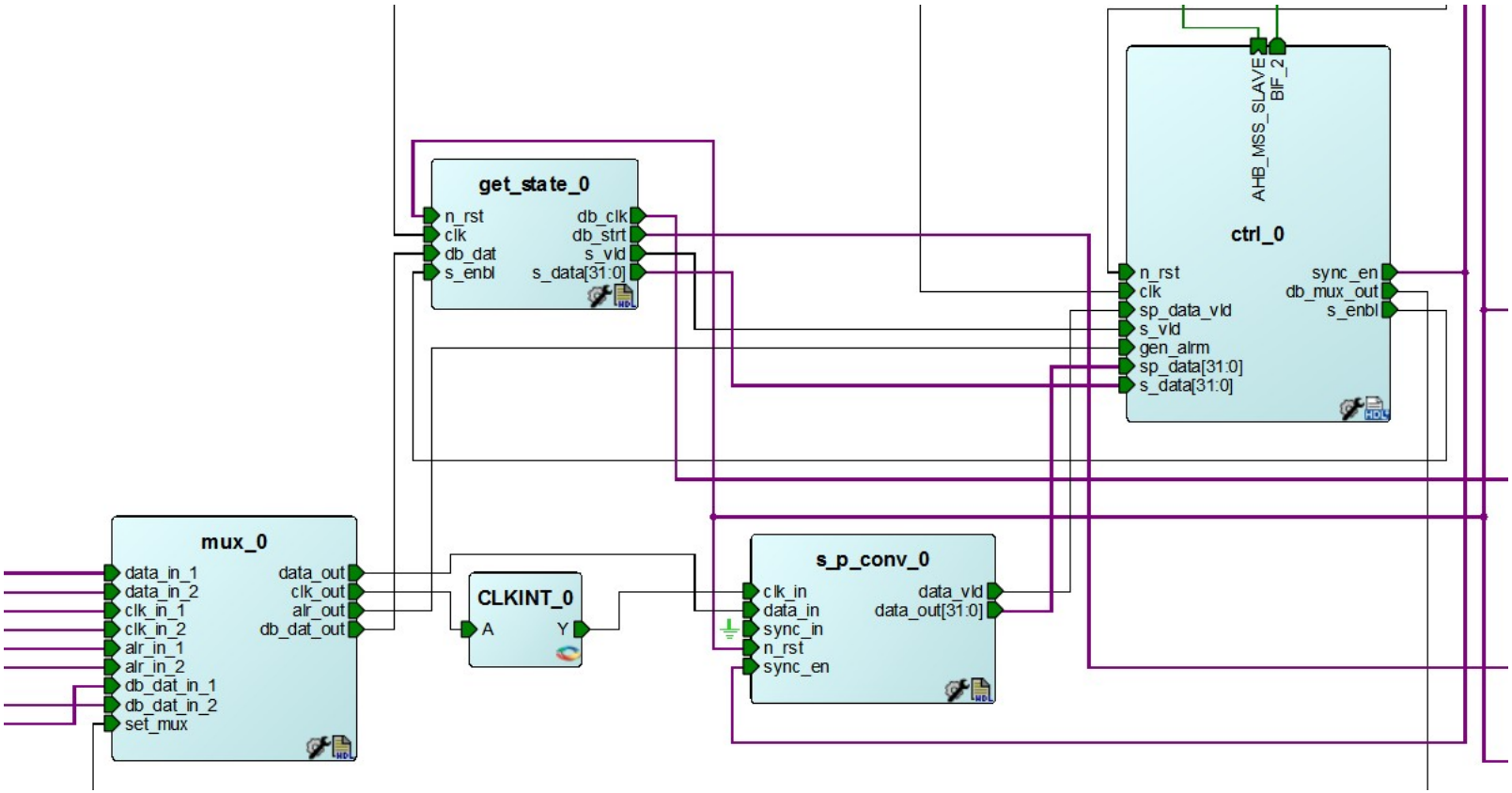
Workflow for the daughter board design

1. Copy the project *pll_trng* to a new folder
2. Open the project
3. Accept IP cores updates (if any)
4. Modify the file *pll_trng.vhd* (add the state registrer and interface)
5. Design the block **Get status** of the motherboard (*get_state.vhd*)
6. Write the testbench and include the two entities (*pll_trng* & *get_state*)
7. Simulate new *pll_trng* and the communication between the two boards using the testbench
8. If OK, define the pinout of the daughter board FPGA
9. Compile the project of the daughter board
10. Program FPGA fabric of the daughter board

Workflow for the motherboard design

1. Open the project
2. Accept IP cores updates (if any)
3. Copy the file get_state from the daughter board project
4. Modify the file *ctrl.vhd* (add new states, e.g. *Get state*...)
5. Modify *mux.vhd* (add new i/f signals)
6. Specify the new pinout of the motherboard FPGA
7. Compile the project of the motherboard
8. Program FPGA fabric of the motherboard

Partial block diagram of the FPGA fabric



Outline

Workflow

1. Copy the script from the project *pll_trng*
2. Modify the script to
   - 2.1 Request to read the TRNG state
   - 2.2 Read the response
   - 2.3 Print out the response (the TRNG state)
3. Run the script
4. Enjoy your new design!

And now...

The End

We hope you enjoyed the tutorial!

Now, it's your turn, please contribute!

Should you need some help, don't hesitate to contact:

Marek Laban (laban@micronic.sk) – for problems concerning the hardware
Marcel Kleja (kleja@micronic.sk) – for problems concerning the firmware
Oto Petura (oto.petura@univ-st-etienne.fr) – for problems concerning the software

”The **HECTOR** project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement number 644052.”

If you need further information, please contact the coordinator:

TECHNIKON Forschungs- und Planungsgesellschaft mbH
Burgplatz 3a, 9500 Villach, AUSTRIA
Tel: +43 4242 233 55, Fax: +43 4242 233 55 77
E-Mail: coordination@hector-project.eu