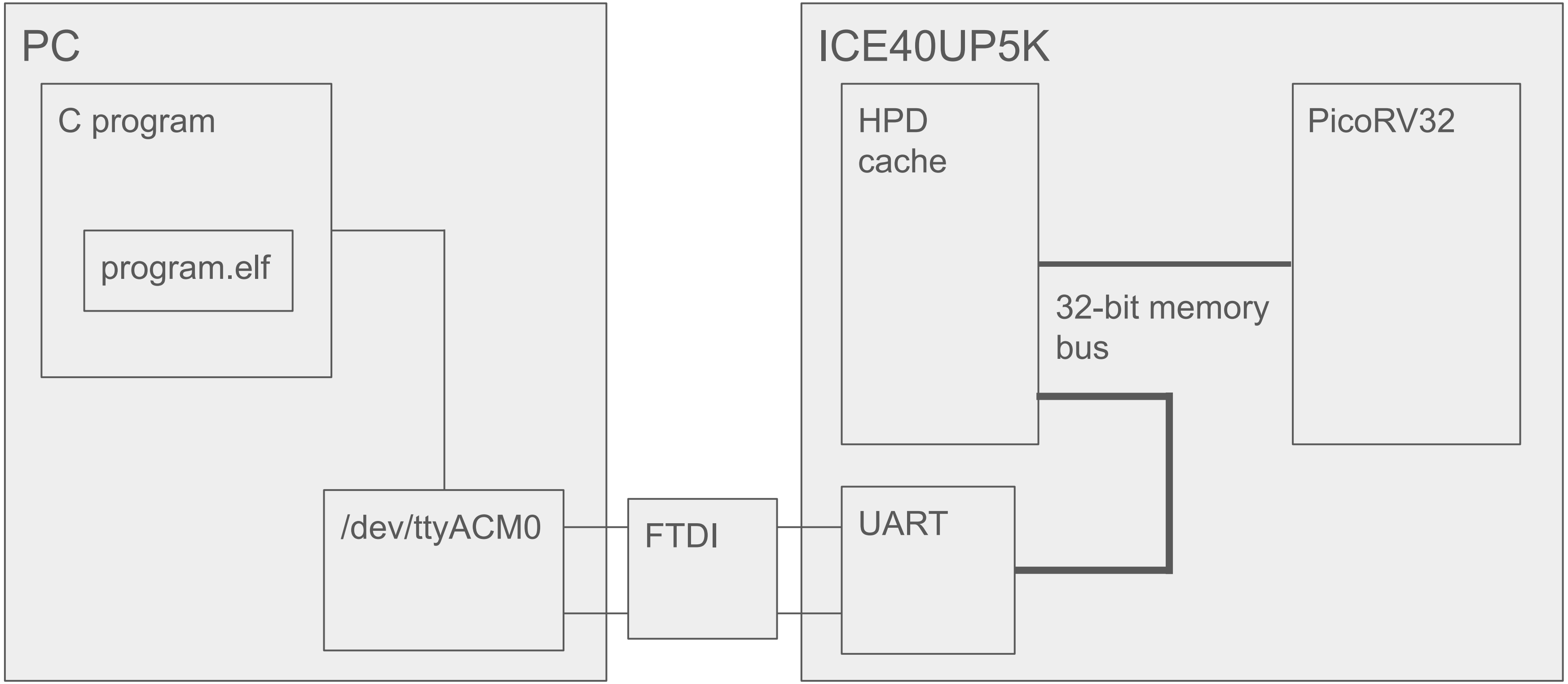
PicoRV memory over UART

Aditya Bedekar, Lennan Tuffy

Project description



UART

Based on the PicoRV's memory requests, send command bytes and either:

- Read: address only
- Write: byte mask, address, and data

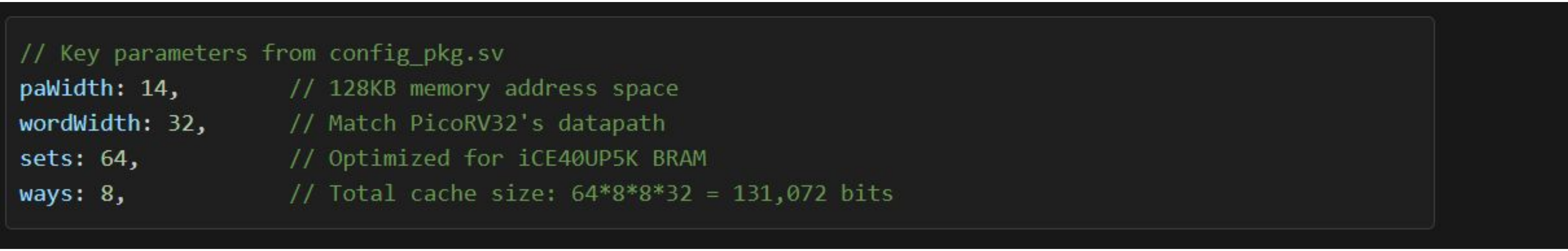
To the computer running the uart controller program.

Used a state machine to choose which bytes to send, similar to what HW1 did

Needed extra modifications after adding the cache in order to communicate using the cache-memory interface

PICORV32

- RISC-V CPU by yosys, works well with low-power FPGAs
- Core: PicoRV32 (4-stage pipeline)
- Target: iCE40UP5K FPGA
- Memory: 128KB (14-bit physical address)



```
// Key parameters from config_pkg.sv
paWidth: 14,          // 128KB memory address space
wordWidth: 32,        // Match PicoRV32's datapath
sets: 64,             // Optimized for iCE40UP5K BRAM
ways: 8,              // Total cache size: 64*8*8*32 = 131,072 bits
```

PICORV32 - continued

- **Pipeline Optimizations**
- MSHR (Miss Status Handling Registers) configured for pipeline stages:
- 4 MSHR sets (one per pipeline stage)
- 2 MSHR ways for concurrent misses
- Write buffer: 8 directory entries, 4 data entries
- Memory Interface
- 32-bit address and data width
- Write-back policy for better performance
- Transaction ID width: 4 bits (16 outstanding transactions)

PICORV32- MSHR

- Purpose: MSHRs track outstanding cache misses to avoid blocking new memory accesses.
- Eg:
- 1) CPU requests data at address 0X1000, cache miss
- 2)MSHR store its request, while allowing CPU to continue
- 3) If another request 0x1004 hits same cache line, MSHR can merge it into the same transaction. In our configuration there are 4 MSHR sets, and 2 MSHR ways which allows for two concurrent cache misses.
- **Write Buffer:** holds write operations before they are sent to main memory.
- 8 directory entries (to track where data goes), 4 data entities (to temporarily store data before writing)
- Transaction ID width: used to identify memory requests

HPD CACHE

- open-source High-Performance, Multi-requester, Out-of-Order L1 Dcache for RISC-V cores and accelerators.
- hpdcache_lint: 32 bit datapath ,single requester, memory interface signals for read and write operations
- Hpdcache_wrapper: adapter module between picorv32 and hpdcache

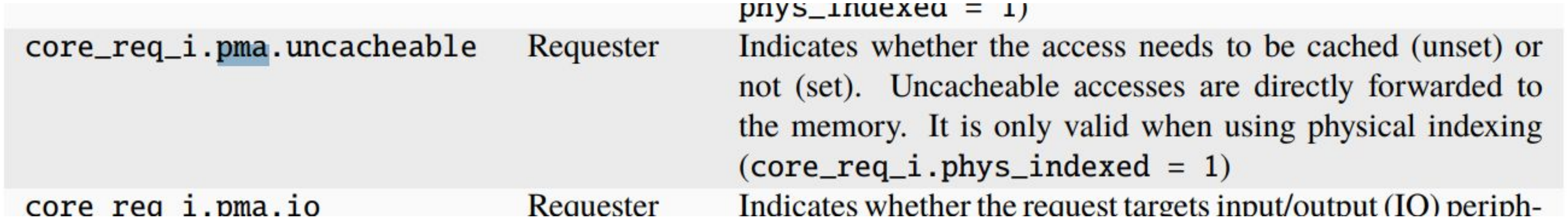
  Cache control signals, write buffer management

- Documentation:- https://github.com/openhwgroup/cv-hpdcache/releases

HPD CACHE - continued

- **PMA:** *An underlying system can have multiple physical memory attributes (PMA)*
- **Cacheability:** For a memory space that is segmented (where segment is defined from base address to end address) some segments are cacheable. The HPD CACHE needs to know which segments are cacheable .
- Eg: for a given read request, should read data be copied into the cache
- Request interface implements an uncacheable bit, when bit is set to 1 , the access is uncacheable.
- Eg: For uncacheable memory, consider I/O device register like a UART serial port.

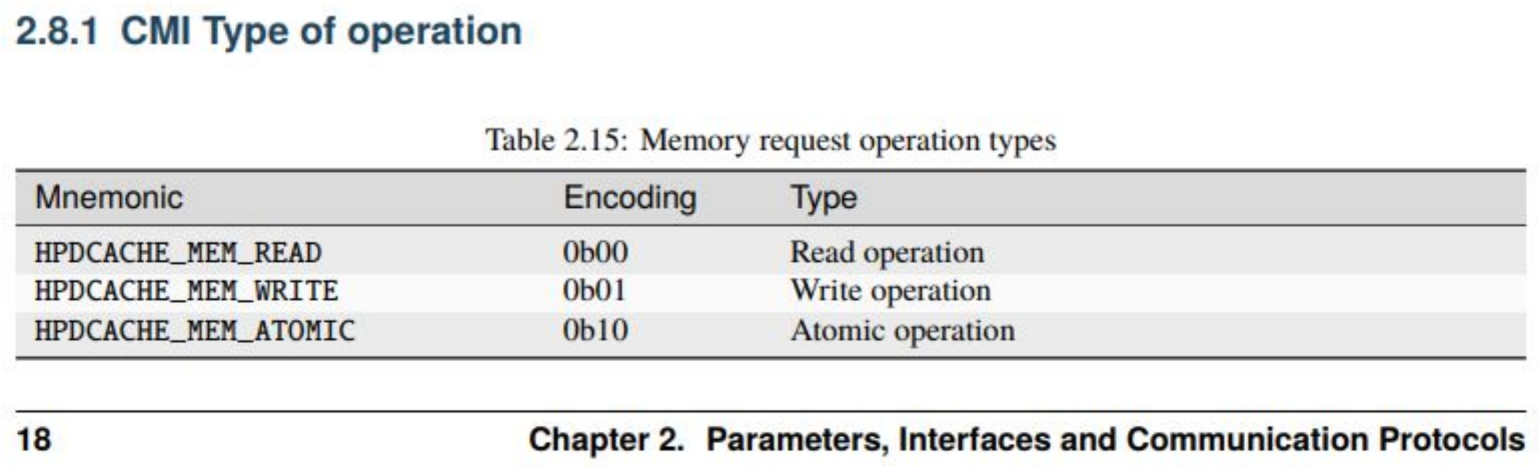
  To ensure latest value is read from hardware



		phys_indexed = 1)
core_req_i.pma.uncacheable	Requester	Indicates whether the access needs to be cached (unset) or not (set). Uncacheable accesses are directly forwarded to the memory. It is only valid when using physical indexing (core_req_i.phys_indexed = 1)
core_req_i.pma.io	Requester	Indicates whether the request targets input/output (IO) periph-

HPD CACHE - continued

- For the cache memory interface (CMI) , these are a few types of memory request operation types that are supported



2.8.1 CMI Type of operation

Table 2.15: Memory request operation types

Mnemonic	Encoding	Type
HPDCACHE_MEM_READ	0b00	Read operation
HPDCACHE_MEM_WRITE	0b01	Write operation
HPDCACHE_MEM_ATOMIC	0b10	Atomic operation

C program and ELF

Install risc-v gnu toolchain -> picorv recommends the version from 2018

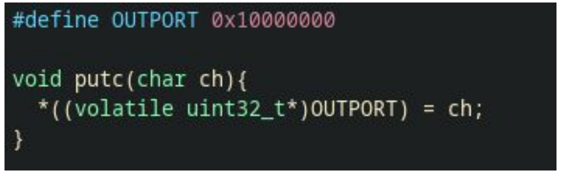
Writing the firmware files in start.S and firmware.c

Compiling and linking to create a .elf file

Reading the .elf file to respond to the core’s requests

- ELF headers - elf header, program header, section headers
- Storing the file contents in an accessible way

Memory map

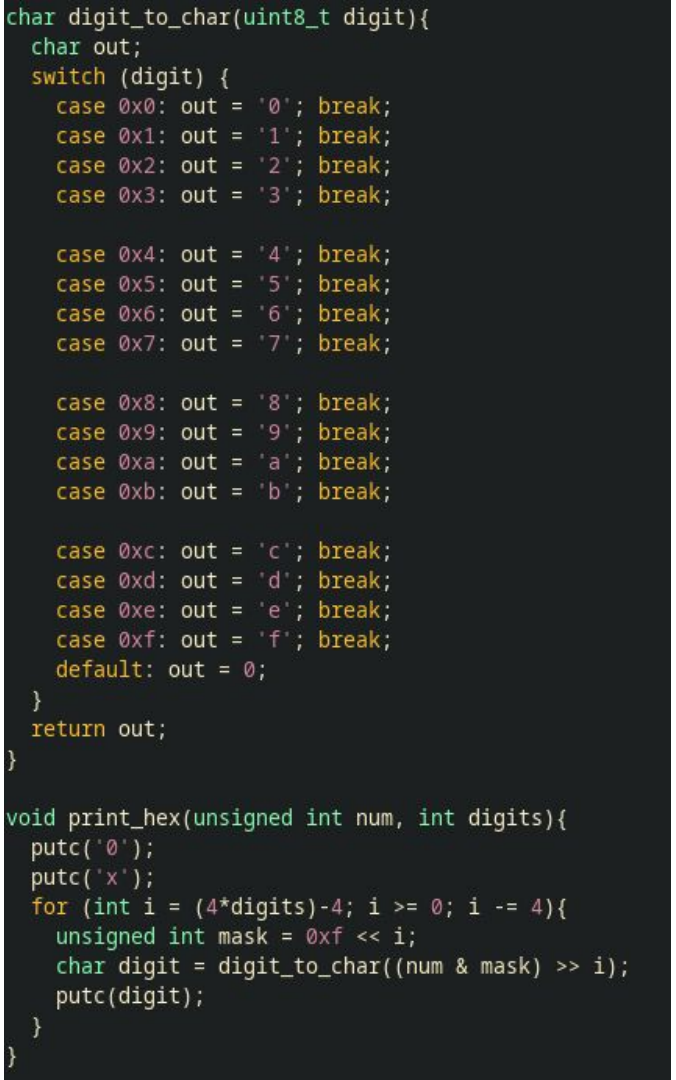

```c
#define OUTPORT 0x10000000

void putc(char ch){
  *((volatile uint32_t*)OUTPORT) = ch;
}
```

The C program interprets writes to 0x10000000 as print statements.

The “virtual” memory doesn’t have space for this address

- Automatically print to stdout if memory is out of bounds


```c
char digit_to_char(uint8_t digit){
  char out;
  switch (digit) {
    case 0x0: out = '0'; break;
    case 0x1: out = '1'; break;
    case 0x2: out = '2'; break;
    case 0x3: out = '3'; break;

    case 0x4: out = '4'; break;
    case 0x5: out = '5'; break;
    case 0x6: out = '6'; break;
    case 0x7: out = '7'; break;

    case 0x8: out = '8'; break;
    case 0x9: out = '9'; break;
    case 0xa: out = 'a'; break;
    case 0xb: out = 'b'; break;

    case 0xc: out = 'c'; break;
    case 0xd: out = 'd'; break;
    case 0xe: out = 'e'; break;
    case 0xf: out = 'f'; break;
    default: out = 0;
  }
  return out;
}

void print_hex(unsigned int num, int digits){
  putc('0');
  putc('x');
  for (int i = (4*digits)-4; i >= 0; i -= 4){
    unsigned int mask = 0xf << i;
    char digit = digit_to_char((num & mask) >> i);
    putc(digit);
  }
}
```

What did we learn?

ELF: writing firmware with riscv-gnu-toolchain, using newlib

Verilator: DPI-C with elf.h - both the c program and tb uses the same file

SystemVerilog language features: structs, sv header files, macros for typedefs

Integration an open-source cache with another open-source risc-v cpu core