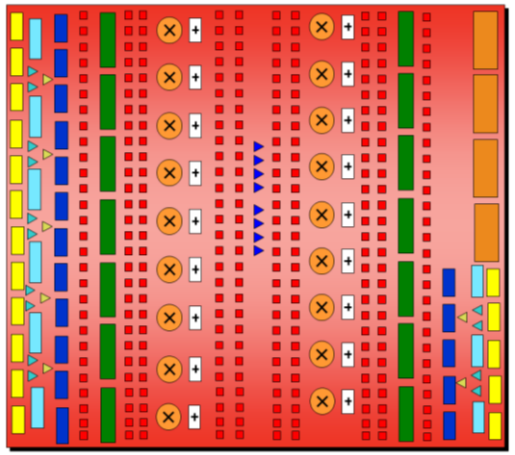
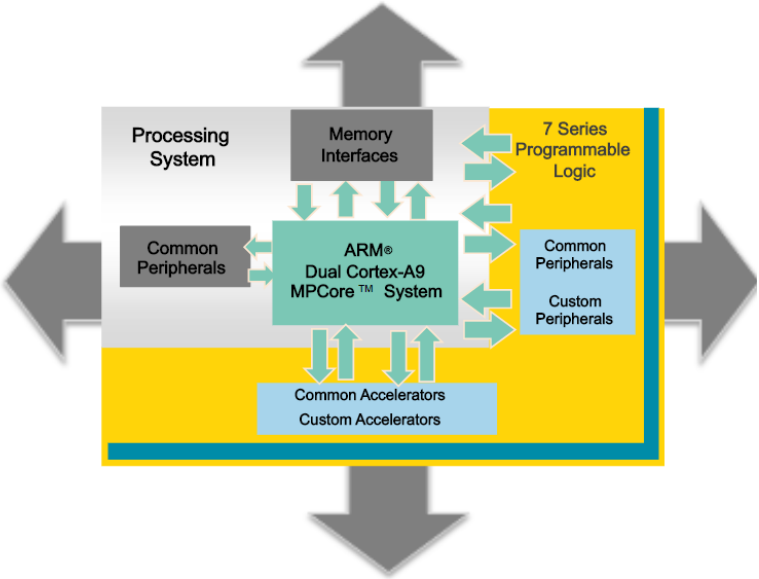
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design





Convolution/Filter Task

- Parallel computing process by nature
- N number of taps/weights/filter coefficients
- N multiplications should happen in parallel

$w_o = 1, w_1 = 2, w_2 = 3$

$x_0 = 1\ then\ y_0 = x_0 w_0 = 1$

$x_1 = 2\ then\ y_1 = x_0 w_1 + x_1 w_0 = 4$

$x_2 = 1\ then\ y_2 = x_0 w_2 + x_1 w_1 + x_2 w_0 = 8$

$x_3 = 2\ then\ y_3 = x_1 w_2 + x_2 w_1 + x_3 w_0 = 10$

Viewed as an Equation

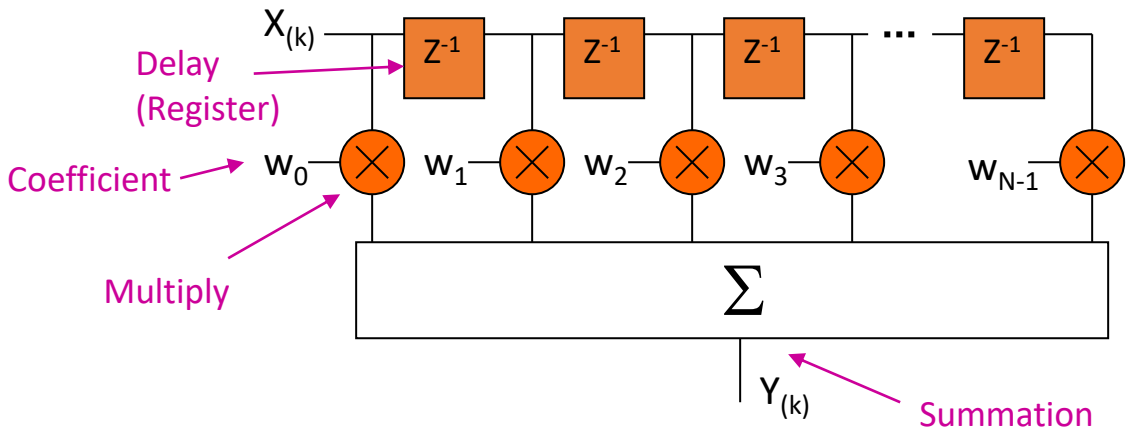
$$Y_{(k)} = \sum_{i=0}^{i=N-1} w_i . X_{(k-i)}$$

Coefficients

Accumulate N times

Multiply

Viewed as a Diagram



Sequential vs. Parallel DSP Processing

SIMD: Single Instruction Multiple data

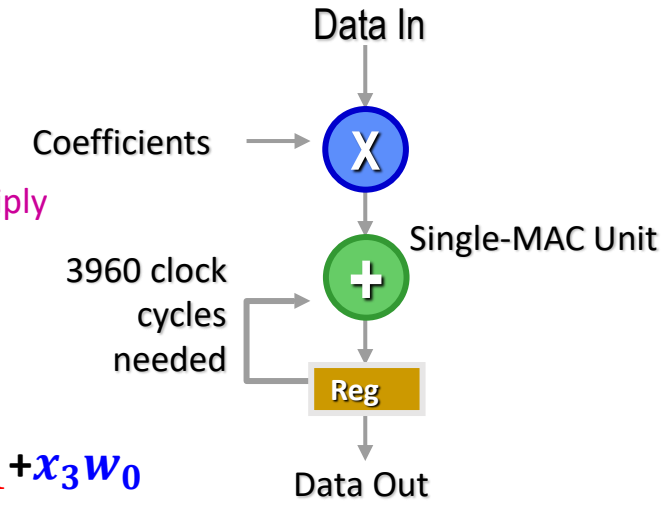
Standard DSP Processor – Sequential (Generic DSP)

$$Y_{(k)} = \sum_{i=0}^{i=N-1} w_i . X_{(k-i)}$$

Coefficients

Accumulate N times

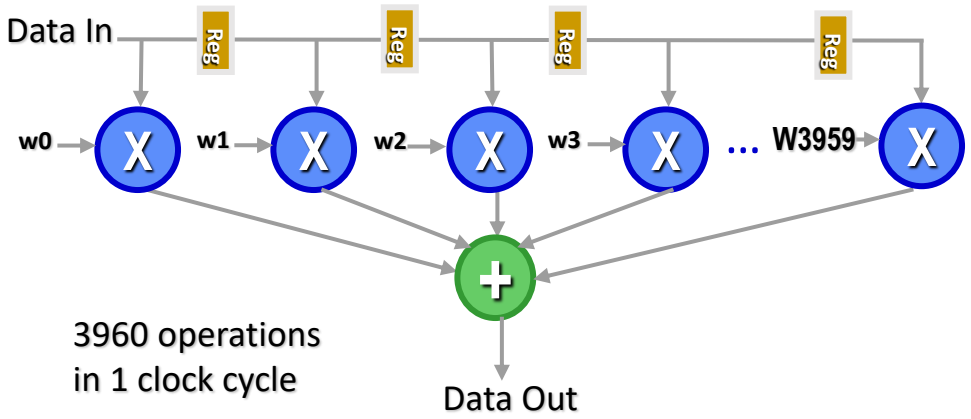
Multiply



$$y_3 = x_1 w_2 + x_2 w_1 + x_3 w_0$$

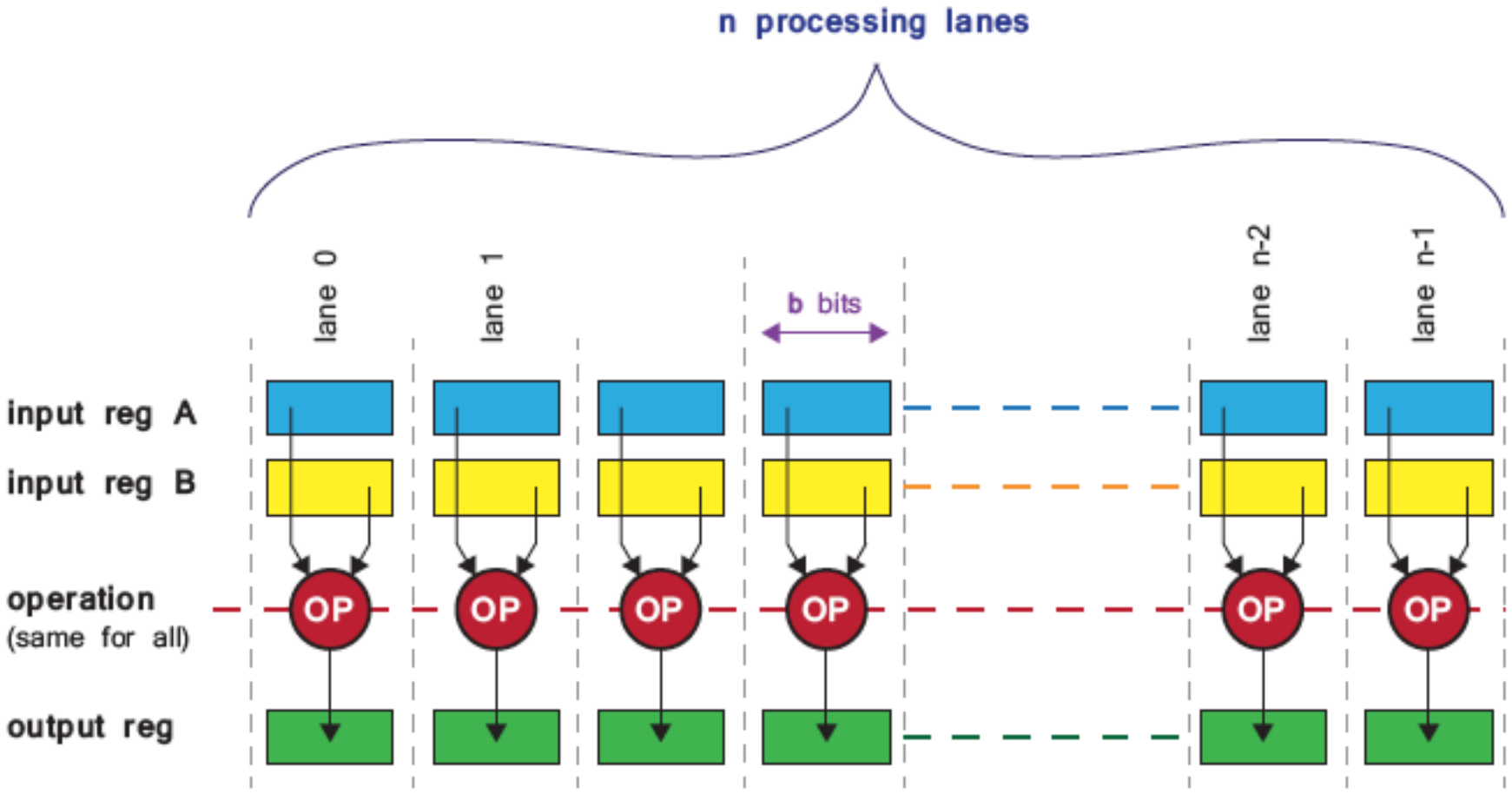
$$\frac{1.2\ GHz}{3960\ clock\ cycles} = 303\ KSPS$$

FPGA - Fully Parallel Implementation (7 Series FPGA)



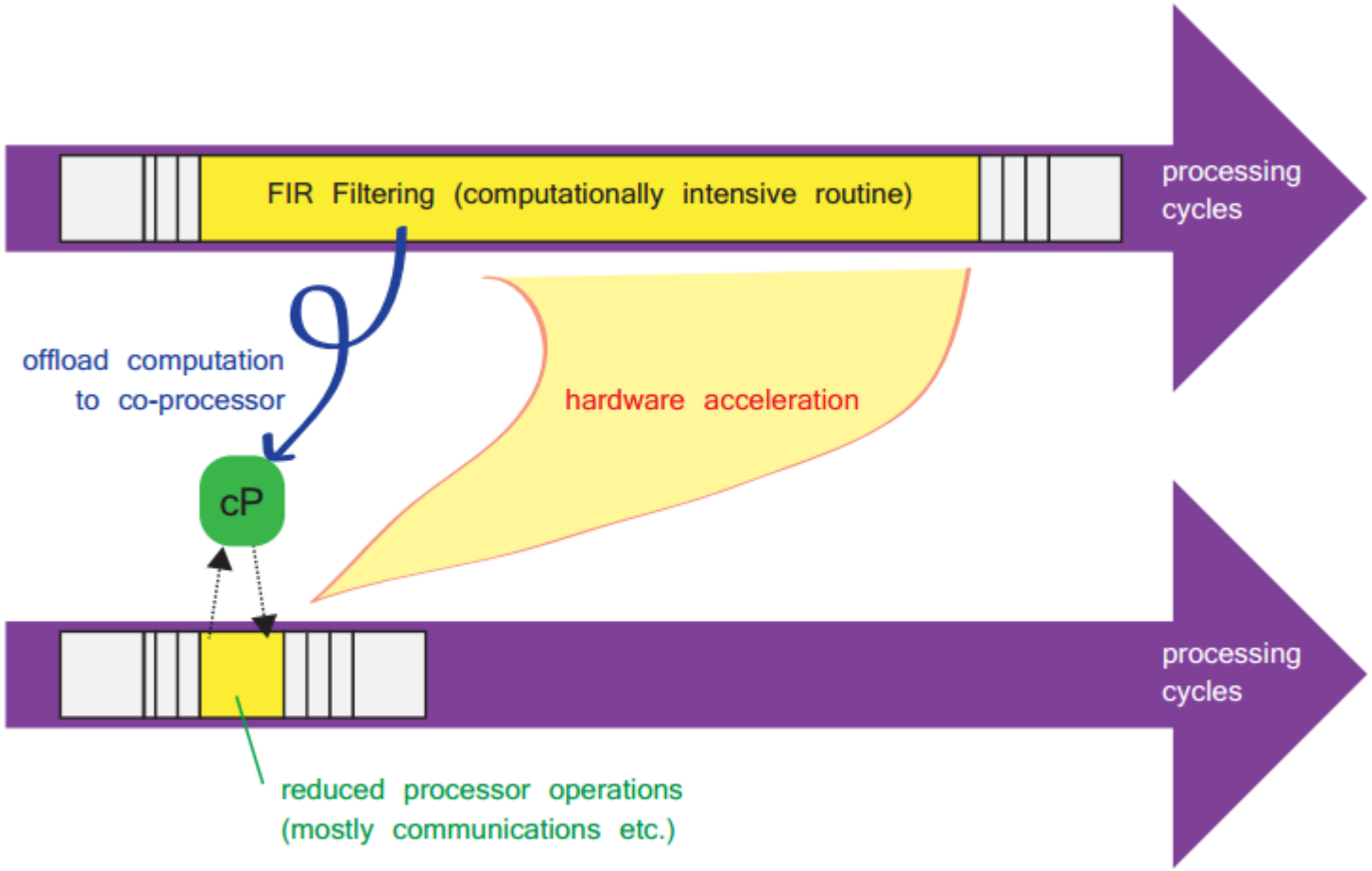
$$\frac{600\ MHz}{1\ clock\ cycle} = 600\ MSPS$$

SIMD (Single Instruction Multiple Data) Processing in the NEON



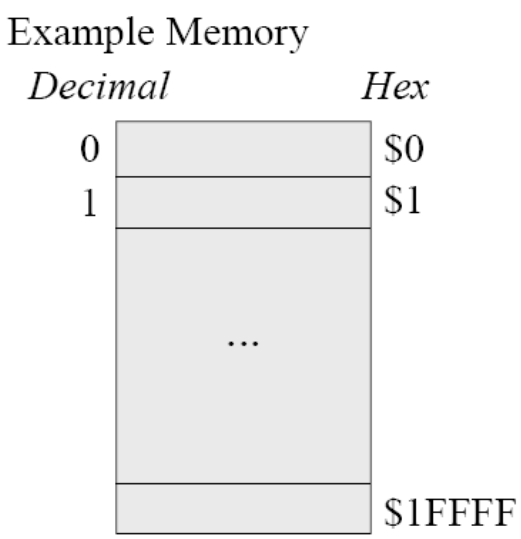
Source: The Zynq Book

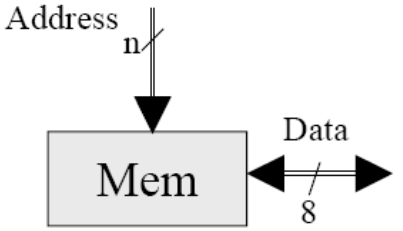
SIMD (Single Instruction Multiple Data) Processing in the NEON



- This means that complex tasks that would require a large number of sequential CPU clock cycles to compute can be executed far quicker using the dedicated coprocessor.

Memory





- 1 address line → 2 data cells
- 2 address line → 4 data cells
- 3 address line → 8 data cells

  …
- n address line → 2^n data cells

- Memory Sizes

1KB = 1024 Byte	Kilo	10 address bits
1 MB = 1024 KB	Mega	20 address bits
1 GB = 1024 MB	Giga	30 address bits

- The number of memory addresses available to CPU is called address space.
- Max. memory size = address space * word length (e.g. 8)

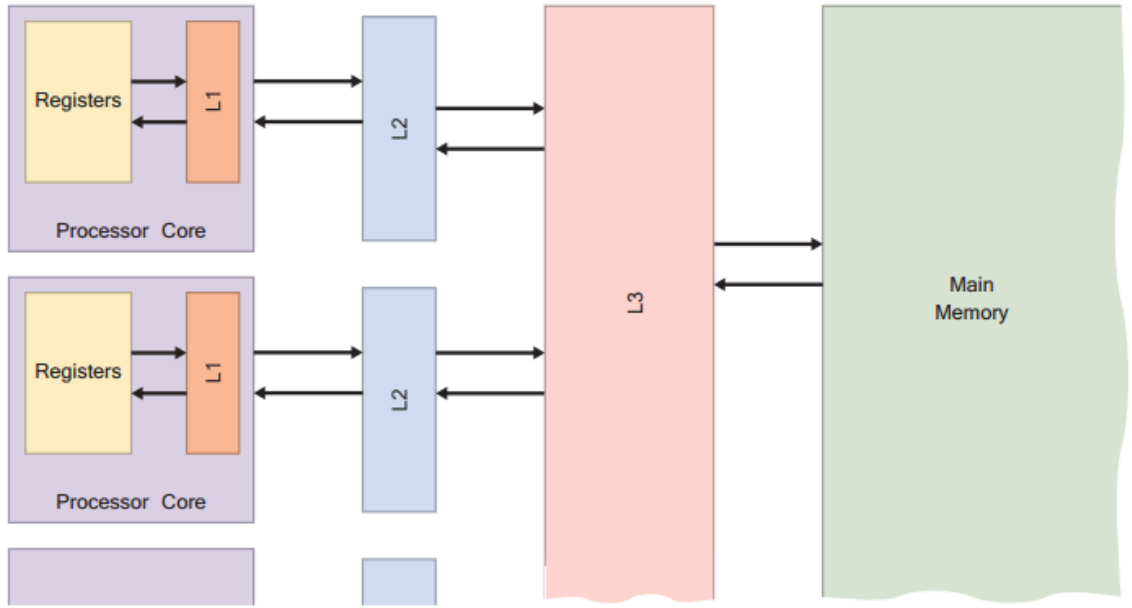
Example: 8 bit controller

- 16 address bits
  → max. 64 KB memory size

Example: 32 bit controller

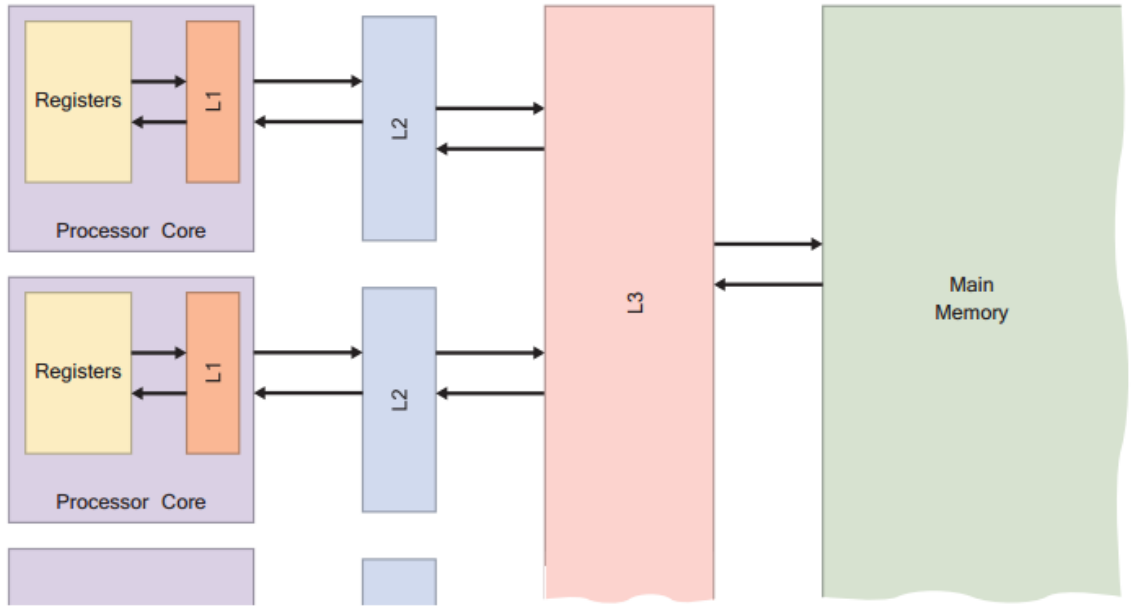
- 24 address bits, addressing single bytes
  (or 23 address bits, addressing 16 bit words)
  → max. 16 MB memory size
- 27 address bits
  → max. 128 MB memory size

Memory



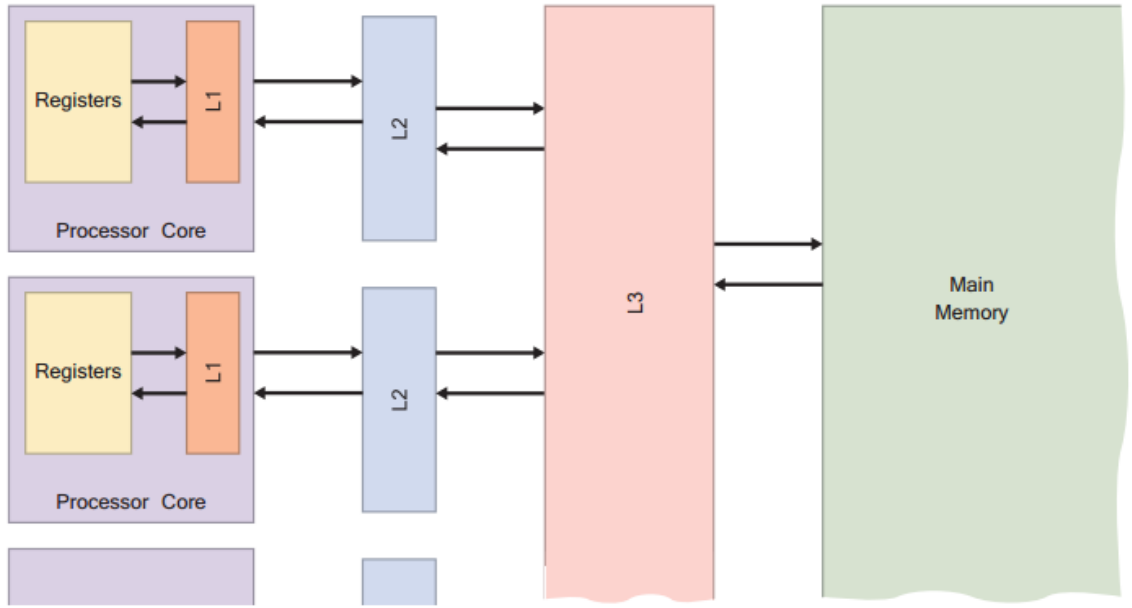
- As processors typically read data at a much faster rate than the system's main memory, the processor speed is constrained to that of the memory.
- To overcome this problem, memory is divided into multiple small chunks known as cache.

Memory



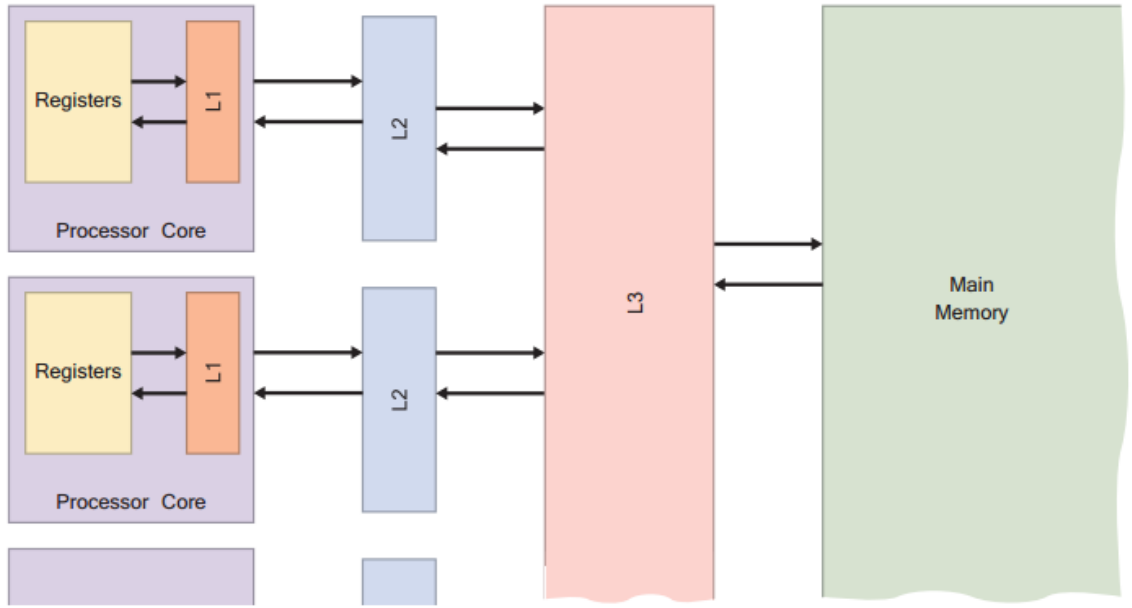
- A cache is a small memory located between the CPU and main memory. It is designed to have lower access time than main memory.
- Cache is used to store data that is frequently accessed by the processor from main memory.
- Operations which make use of cached data are therefore much faster than those where the data is in main memory only.

Memory



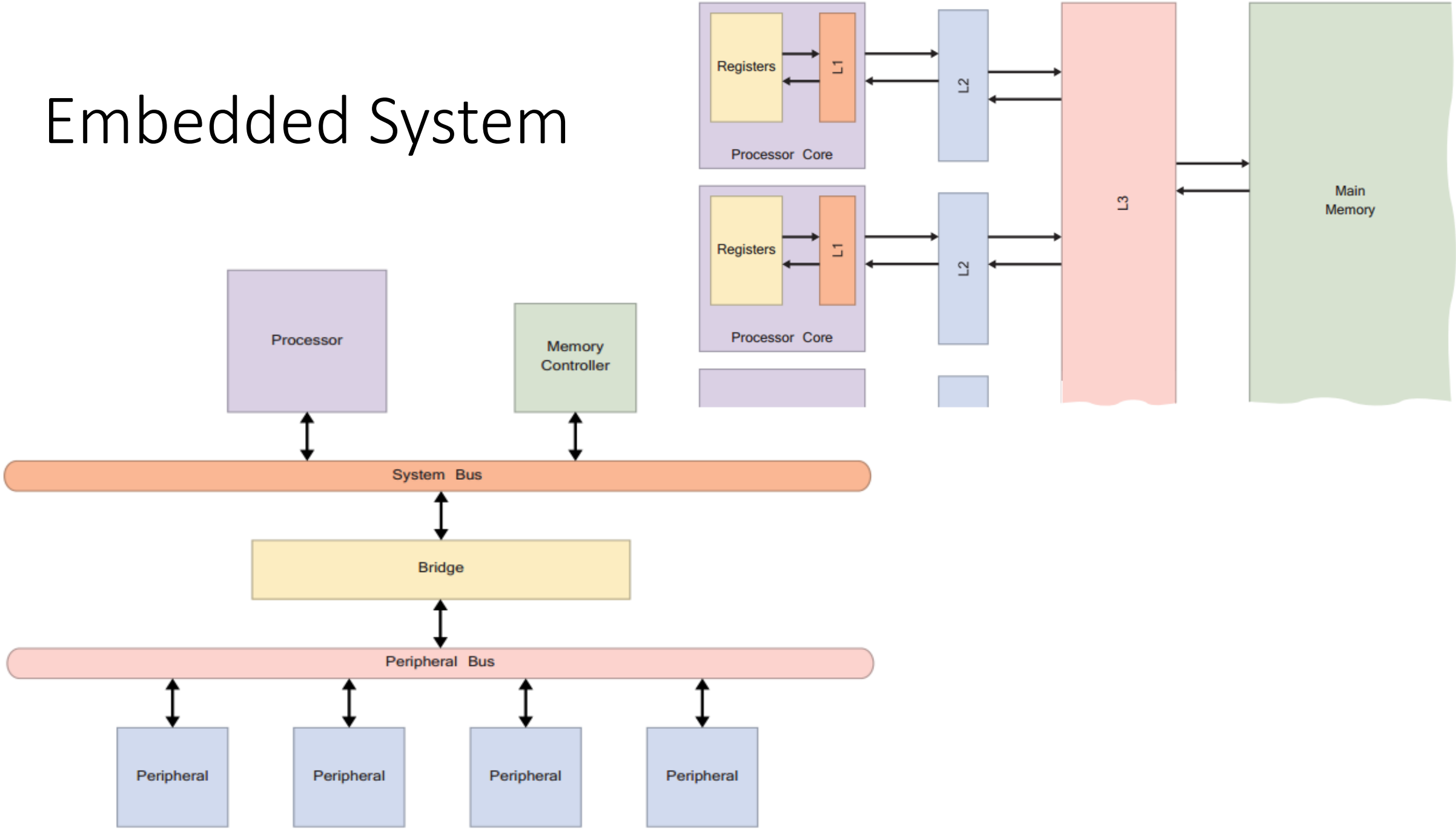
- By including cache memory in a system (which contains the most frequently accessed data that can be read at a higher rate than main memory), the processor is no longer constrained to the speed of the main memory.
- This leads to an increased efficiency in terms of data access.

Memory



- The processor speed is still limited, however, in the event of a cache miss — whereby the processor fails in its attempt to read or write data in the cache.
- This results in the data being read/written to main memory, and increased access latency

Embedded System

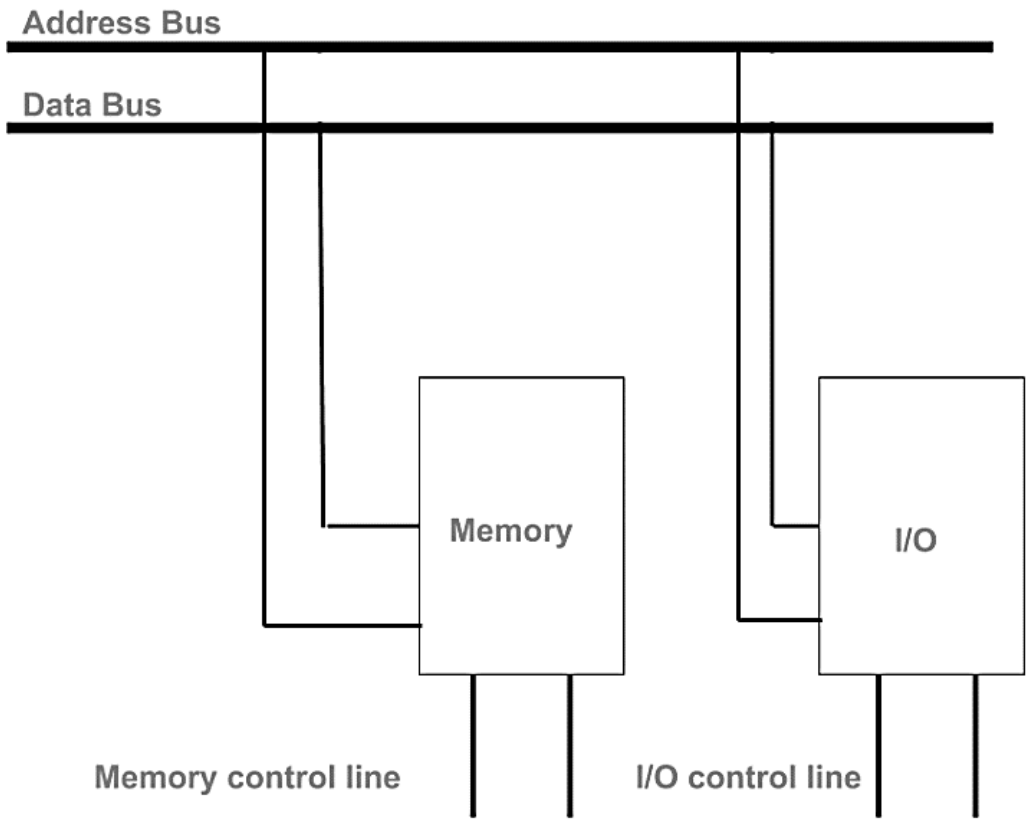


Memory mapped I/O and Isolated I/O

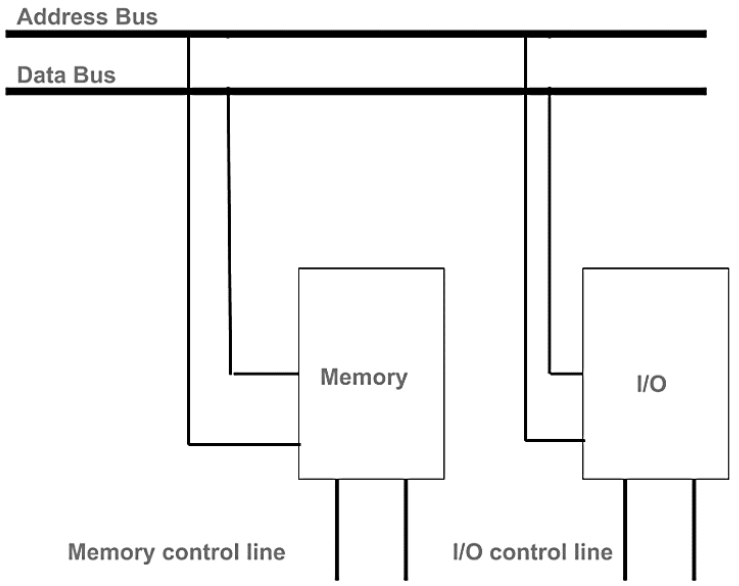
- As a CPU needs to communicate with the various memory, and peripherals/input-output devices (I/O), there are three ways in which the buses can be allotted to them :
  1. Separate set of address, control and data bus to I/O and memory. (System bus and peripheral bus)
  2. **Isolated IO:** Have common bus (data and address) for I/O and memory but separate control lines (only System bus)
  3. **Memory mapped IO:** Have common bus (data, address, and control) for I/O and memory (only System bus)

Isolated I/O

- **Isolated IO**: Have common bus (data and address) for I/O and memory but separate control lines. (only System bus)

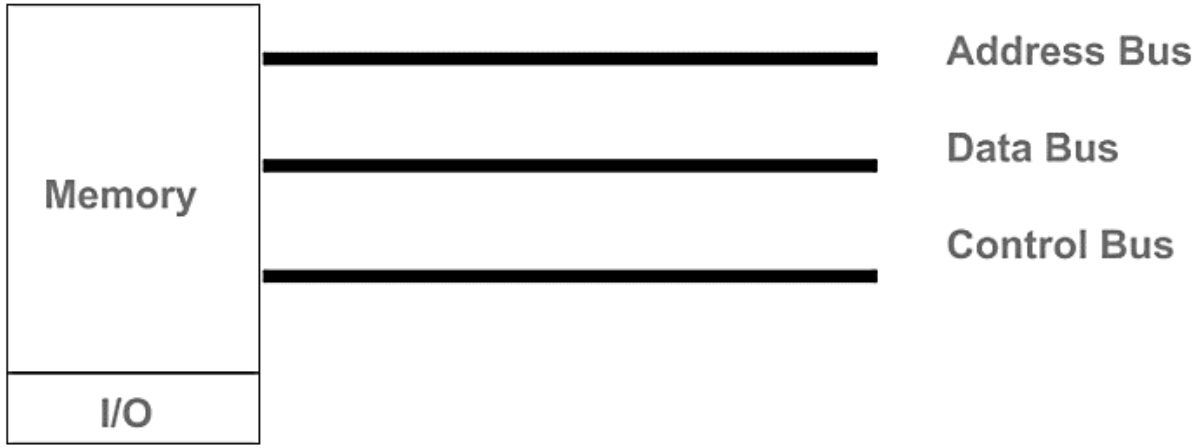


Isolated I/O



- **Isolated IO:** Have common bus (data and address) for I/O and memory but separate control lines. (only System bus)
- When CPU decodes instruction and if data is for I/O, it places the address on the address line and set I/O read or write control line ON due to which data transfer occurs between CPU and I/O.
- Address space of memory and I/O is isolated.
- Different read-write instructions for both I/O and memory.

Memory Mapped I/O



- **Memory mapped IO**: Have common bus (data, address, and control) for I/O and memory. (only System bus)
- In this case every bus in common due to which the same set of instructions work for memory and I/O.
- Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.

ISOLATED I/O	MEMORY MAPPED I/O
Memory and I/O have seperate address space	Both have same address space
All address can be used by the memory	Due to addition of I/O addressable memory become less for memory
Separate instruction control read and write operation in I/O and Memory	Same instructions can control both I/O and Memory
Larger in size due to more buses	Smaller in size
It is complex due to separate separate logic is used to control both.	Simpler logic is used as I/O is also treated as memory only.

Programmer's View of Custom Accelerators & Peripherals

Start Address	Description
0x0000_0000	External DDR RAM
0x4000_0000	Custom Peripherals (Programmable Logic including PCIe)
0xE000_0000	Fixed I/O Peripherals
0xF800_0000	Fixed Internal Peripherals (Timers, Watchdog, DMA, Interconnect)
0xFC00_0000	Flash Memory
0xFC00_0000	On-Chip Memory

Start Address	Description
0x4000_0000	Accelerator #1 (Video Scaler)
0x6000_0000	Accelerator #2 (Video Object Identification)
0x8000_0000	Peripheral #1 (Display Controller)

Code Snippet

```
int main() {

int *data = 0x1000_0000;
int *accel1 = 0x4000_0000;

// Pure SW processing
Process_data_sw(data);

// HW Accelerator-based processing
Send_data_to_accel(data, accel1);
process_data_hw(accel1);
Recv_data_from_accel(data, accel1);
}
```

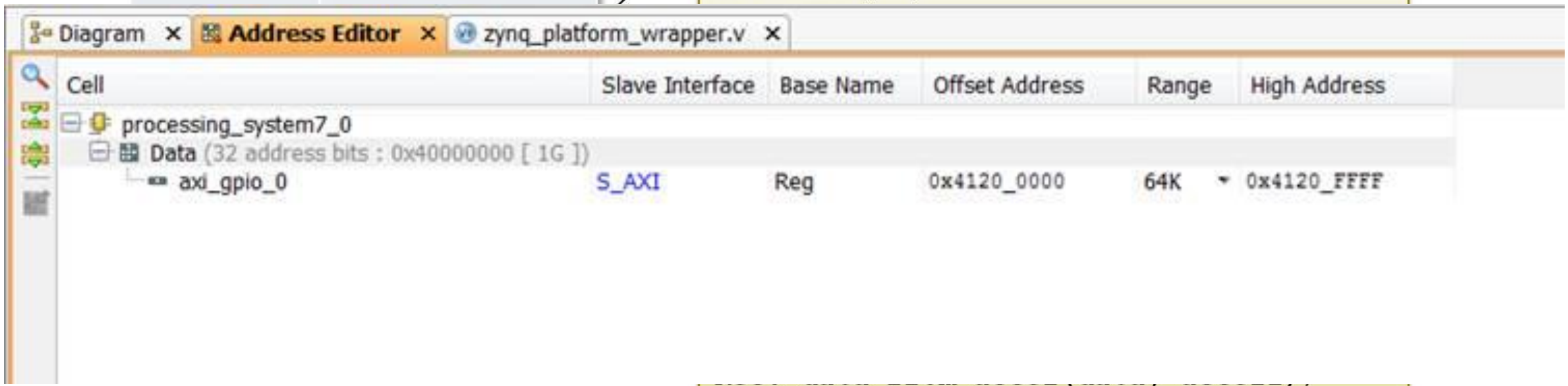
Programmer's View of Custom Accelerators & Peripherals

Start Address	Description
0x0000_0000	External DDR RAM
0x4000_0000	Custom Peripherals (Programmable Logic including PCIe)

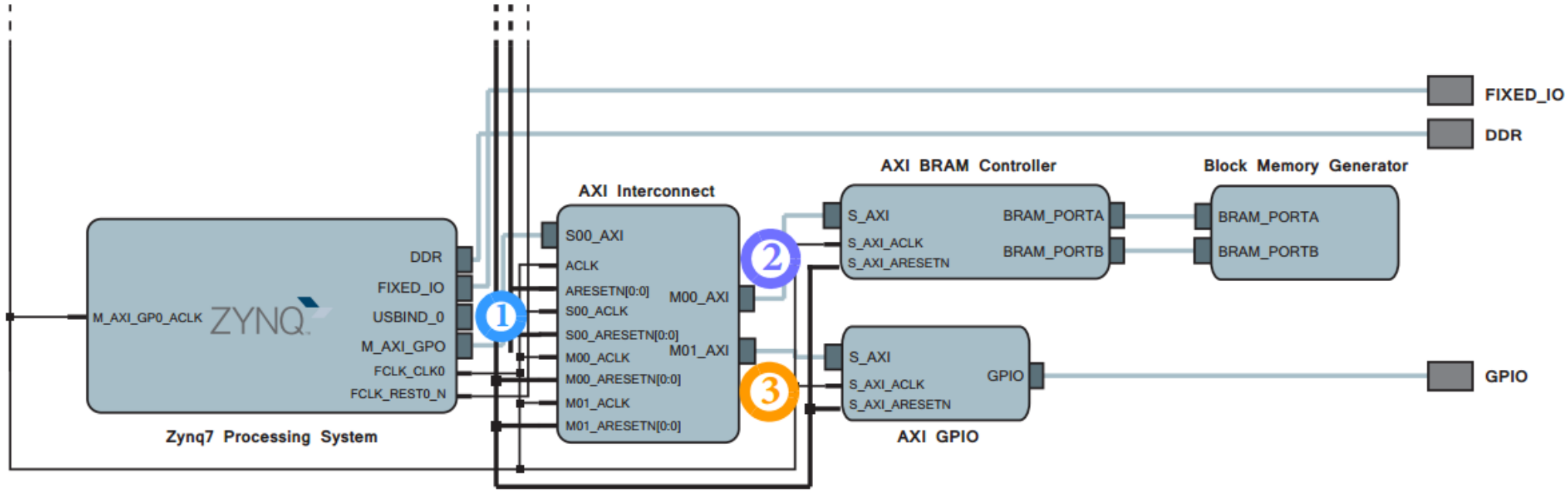
Start Address	Description
0x4000_0000	Accelerator #1 (Video Scaler)
0x6000_0000	Accelerator #2 (Video Object Identification)
0x8000_0000	Peripheral #1 (Display Controller)

Code Snippet

```
int main() {
```




```
}
```

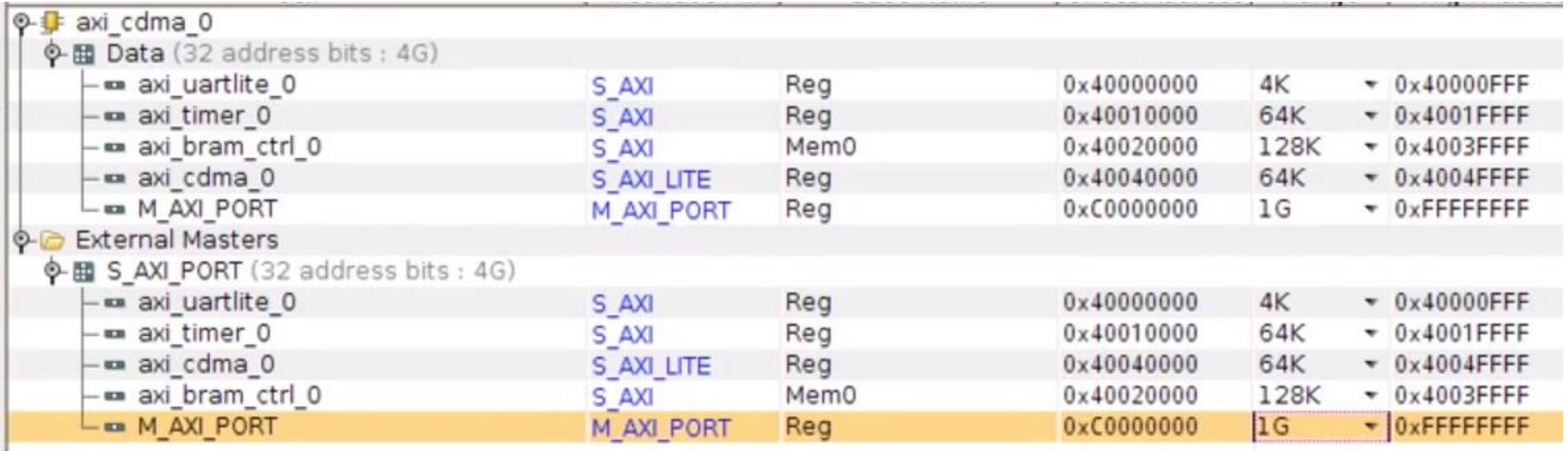


1 The AXI master signal from the Zynq processing system connects to the AXI slave port of the AXI Interconnect block

2 An AXI master signal from the AXI Interconnect connects to the AXI slave port of the BRAM controller

3 An AXI master signal from the AXI Interconnect connects to the AXI slave port of the GPIO instance

1KB = 1024 Byte	Kilo	10 address bits
1 MB = 1024 KB	Mega	20 address bits
1 GB = 1024 MB	Giga	30 address bits



Address Allocations