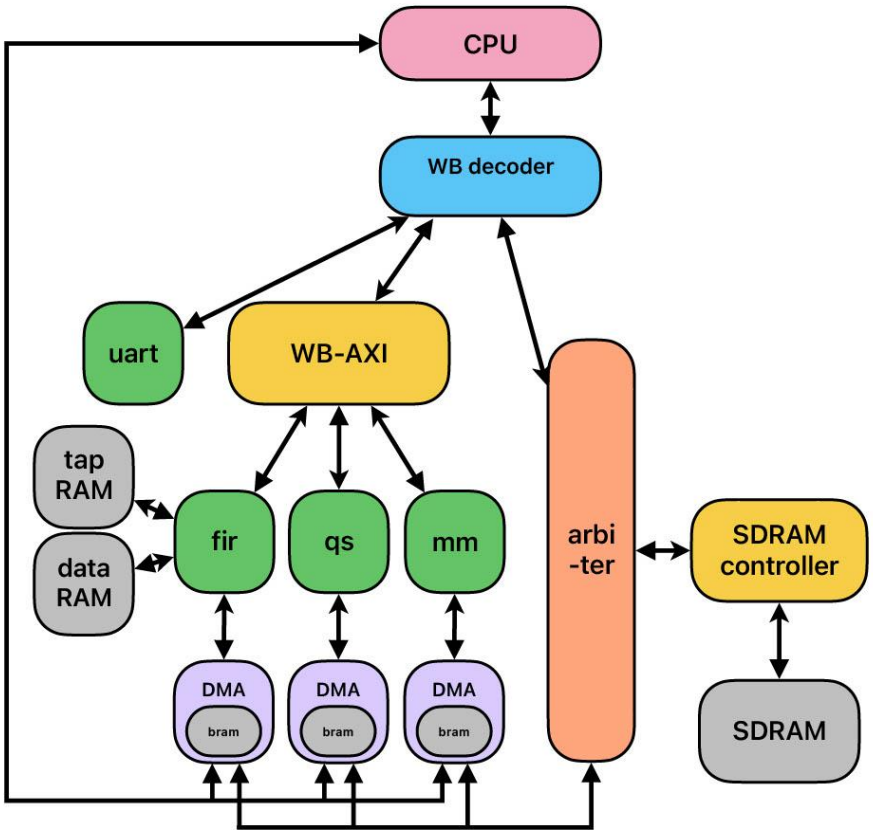
Final Project Report

Group 7　R12943006 謝郡軒
R12943012 蔡東翰
R12943031 李允恩

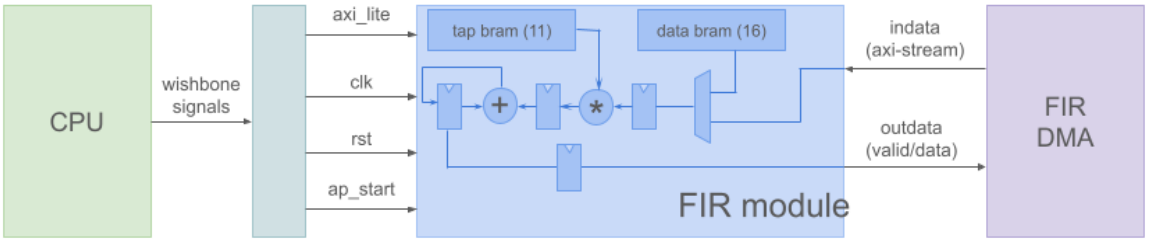
Block Diagram



- In this project, we implemented FIR, QS, MM hardware, and we also added DMA and arbiter to accelerate the calculation, so that hardware can directly communicate with SDRAM.
- Steps
    - CPU sends coefficients to FIR.
    - FIR, QS, and MM start simultaneously.
    - CPU sends address and length of data and answer to DMA.
    - DMA starts reading SDRAM through the arbiter and saves data into data BRAM.
    - When DMA finishes reading data, it can start sending data to hardware, and then hardware starts calculating.
    - Hardware sends answers to DMA, and DMA saves data into answer BRAM.
    - After DMA gets all answers, DMA sends answers back to SDRAM through the arbiter.

FIR

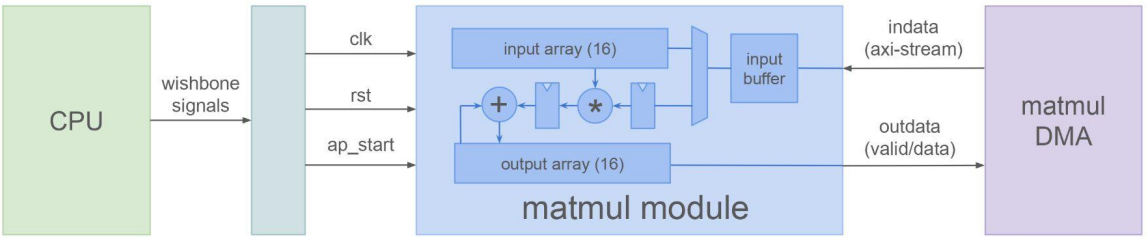
Block diagram



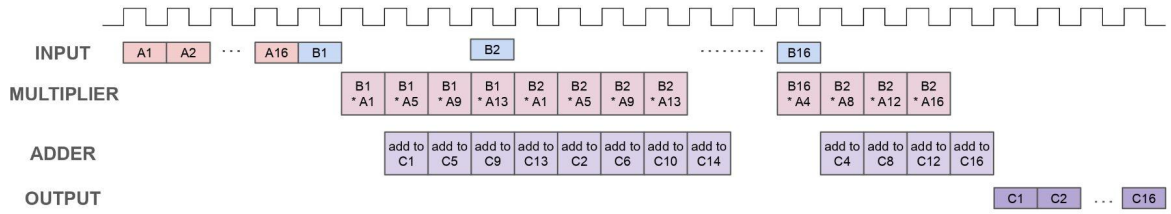
- Use only 1 multiplier and 1 adder.
- CPU first sends coefficients, and FIR saves coefficients in BRAM. After that, DMA sends input data.
- FIR receives new data when the previous answer is outputted.
- FIR produces 1 data per 12 cycles.
- If DMA is fast enough to send and receive the input/output data, the FIR process in our design totally costs about 132 cycles (excluding coef receiving time).

Matmul

Block diagram



Schedule

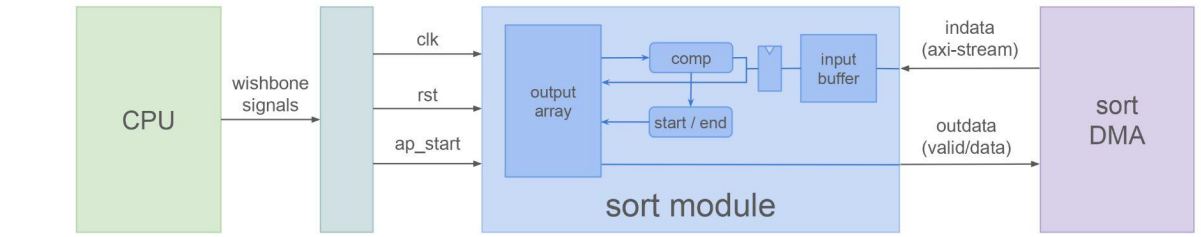


- Use only 1 multiplier and 1 adder.
- After we read the 4*4 A matrix, we read the elements in B matrix one by one and calculate all the results related to the current B element before asking for the next B element, we don't need to save the whole B matrix in our design so therefore we can save about 14 registers.

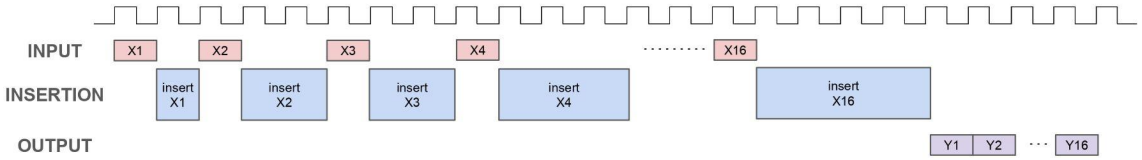
- If DMA is fast enough to send and receive the input/output data, the matrix multiplication process in our design totally costs about 16 (receiving A) + 64 (receiving B and calculating C) + 16 (returning C) = 96 cycles.

Quick Sort

Block diagram



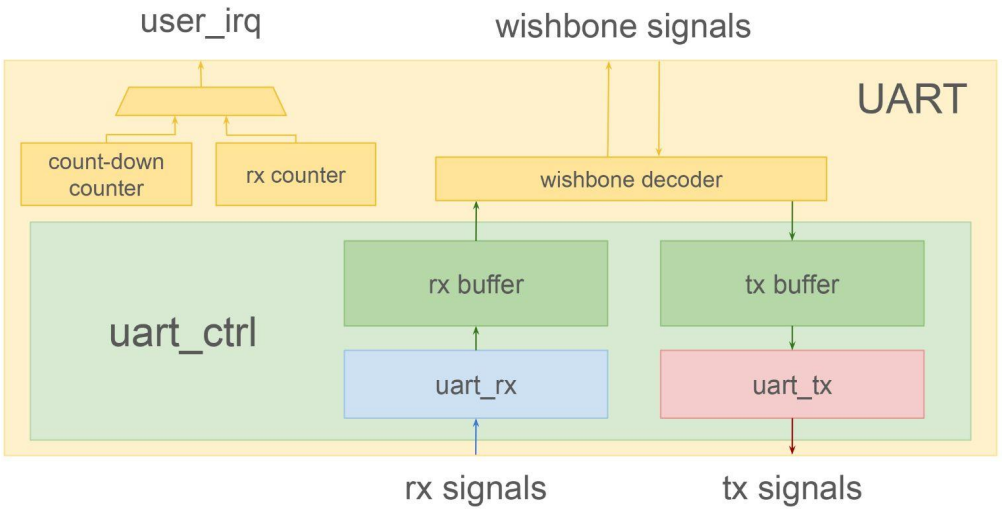
Schedule



- Use only 1 comparator.
- Although quicksort may be more efficient and powerful in C++, however, in this project, we use verilog code for coding and the input sequence comes in serial, so we think using insertion sort will be more suitable.
- Because insertion sort is also in-place, we can use about only 12 registers to sort 10 elements (2 more for buffering).
- If DMA is fast enough to send and receive the input/output data, the insertion sort process in our design totally costs about 53 cycles.

UART

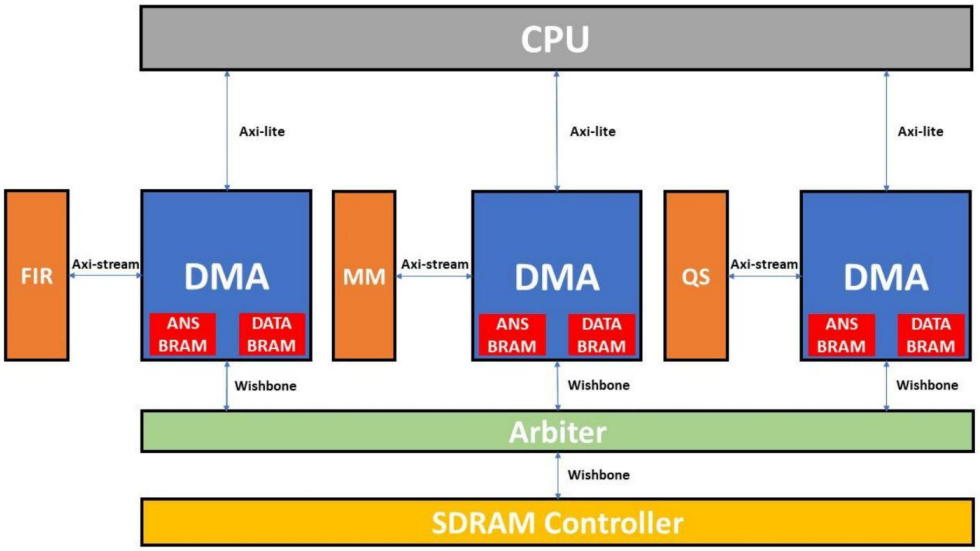
Block diagram



- We add buffers at both rx and tx sides, so that (1) we don't need to enter ISR as soon as one interrupt happens, and (2) we can send another data to be transmitted without waiting for the previous one to finish its transmission.
- Besides, to prevent unsettled interrupts, we have a count-down counter to check if the rx buffer is empty periodically.
- Therefore, we can set two parameters n and m, which the UART generates an irq signal if there has been total n unsettled interrupts or m cycles since the last time entering ISR.

Direct Memory Access

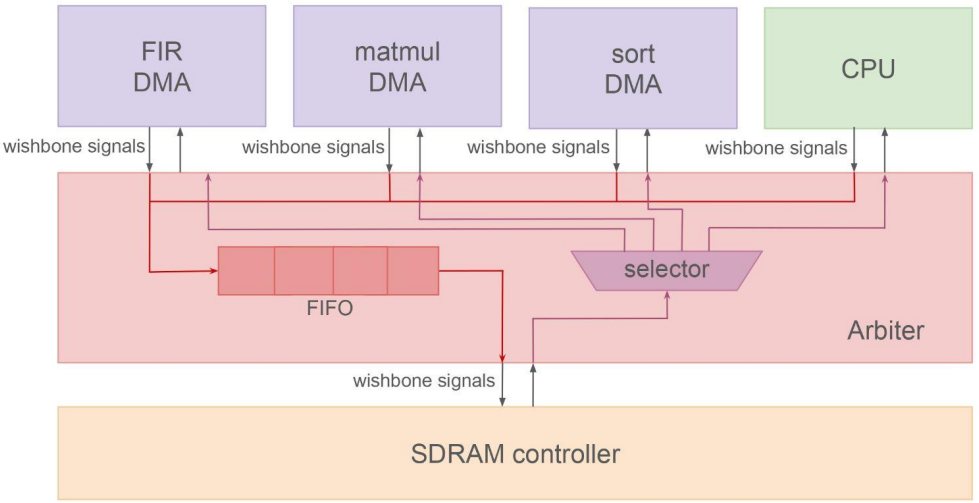
Block diagram



- There are 3 DMAs for FIR, MM, and QS respectively.
- Each of them will receive five data through axi-lite protocol from CPU, including information of data_len, data_addr, ans_len, ans_addr, ap_start, and DMAs have to return the ap_done signal back to CPU when necessary.
- After receiving all data above, DMAs will ask for input data for FIR/MM/QS engines through wishbone protocol from SDRAM controller according to the given data_addr and data_len, and save them into DATA BRAM in order.
- However, we need an arbiter between DMAs and SDRAM controller to avoid collision of data requests.
- After saving all input data into DATA BRAM, DMAs will start sending input data into FIR/MM/QS engines through axi-stream protocol with a view to activating the corresponding computations.
- When FIR/MM/QS engines finish the operations, they will send the answers back to DMAs through axi-stream protocol, which will be saved in ANS BRAM in order by DMAs.
- After saving all answer data into ANS BRAM, DMAs will start sending answer data to arbiter through wishbone protocol. Eventually, arbiter and SDRAM controller will save those data into SDRAM properly.

Arbiter

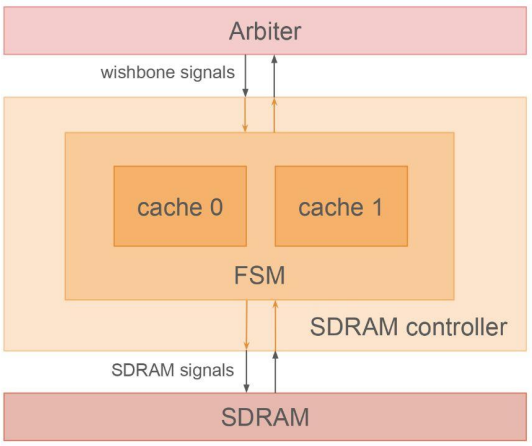
Block diagram



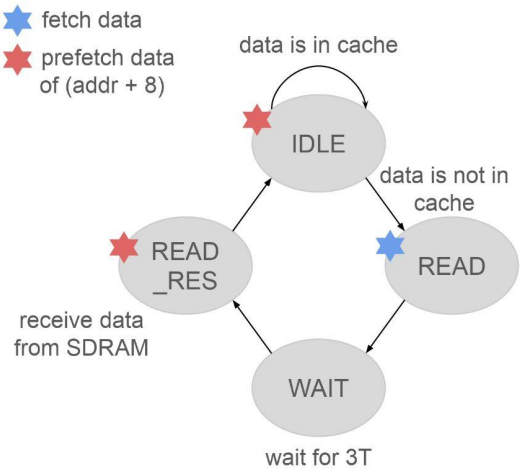
- Arbiter collects the wishbone signals of DRAM access from 3 DMAs and CPU together, putting them in a FIFO, and then arbiter will access the DRAM according the output of FIFO and send back the corresponding wishbone signals to DMA.
- Since wishbone signals cannot be pipelined, arbiter will receive a new request from one source only when the last request from the same source has been fulfilled. Therefore, we only need a FIFO of size 4 to accommodate all the requests from 4 different sources.
- To accelerate the whole process, if two requests comes in at the same time, the priority will be CPU > FIR > matmul > sort

SDRAM Controller

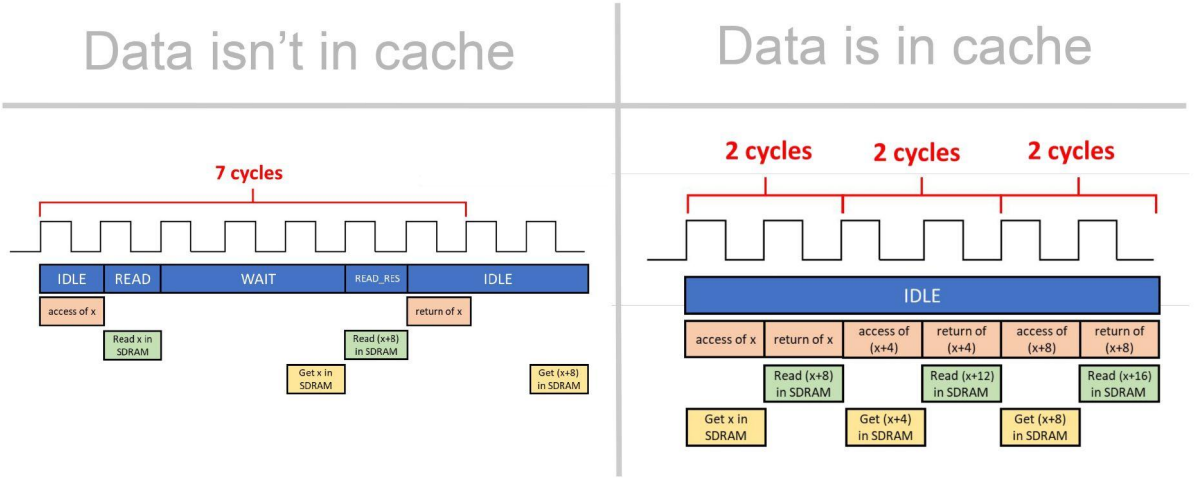
Block diagram



FSM (our different part)



Schedule



- We observe that if this cycle CPU is accessing address "x" then the next cycle CPU will likely access address "x+4".
- Since it costs 3 cycles from we send a request to SDRAM to we get the data, we need to have at least 2 caches to store the prefetch datas.
- Therefore, we use 2 caches where cache n (n = 0, 1) saves the data whose addr[2] == n, and every time we send back a data of address "x" to arbiter we will immediately go prefetch the data of address "x+8" and save it in the corresponding cache.
- At the best situation (we've guessed the prefetch address correctly), the latency of accessing a data from SDRAM will be shortened to 2 CYCLES.

Simulation Result

Compile optimization: O3

```
LA Test uart started
tx data bit index 0: 1
tx data bit index 1: 1
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 0
tx data bit index 5: 0
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 1
Calculation start, Time = 1.46244 ms
Calculation finish, Time = 1.53484 ms
LA Test fir started
correct answer: 0000
correct answer: fff6
correct answer: ffe3
correct answer: ffe7
correct answer: 0023
correct answer: 009e
correct answer: 0151
correct answer: 021b
correct answer: 02dc
correct answer: 0393
correct answer: 044a
LA Test mm started
correct answer: 003e
correct answer: 0044
correct answer: 004a
correct answer: 0050
correct answer: 003e
correct answer: 0044
correct answer: 004a
correct answer: 0050
correct answer: 003e
correct answer: 0044
correct answer: 004a
correct answer: 0050
correct answer: 003e
correct answer: 0044
correct answer: 004a
correct answer: 0050
```

```
LA Test qsort started
correct answer: 0028
correct answer: 037d
correct answer: 09ed
correct answer: 0a6d
correct answer: 0ca1
correct answer: 10ab
correct answer: 120e
correct answer: 1631
correct answer: 1787
correct answer: 2371
LA Test 2 passed
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 1
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 2
rx data bit index 0: 1
rx data bit index 1: 1
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 0
rx data bit index 5: 0
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  15
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  61
```

- mprj == AB00_0000
  - Calculation start
- mprj == AB10_0000
  - Calculation finish

```
reg_mprj_datal = 0xAB000000;
int* temp = all();
reg_mprj_datal = 0xAB100000;
check_ans();
```

Hardware Result

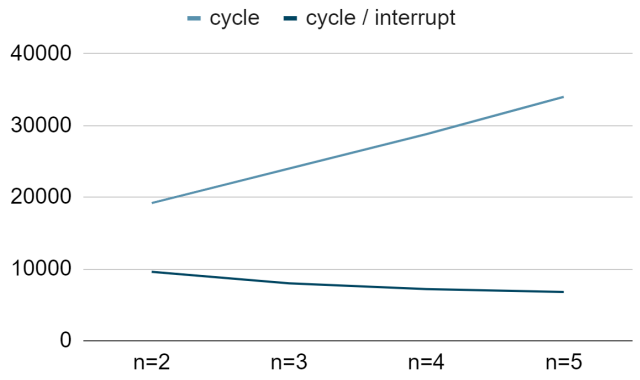
- Original (Lab6)
    - Setup: 2.93 ms
    - Calculation: 4.81(fir) - 0.98(uart) + 2.95(mm) - 0.97(uart) + 0.77(qs) = 6.58ms
    - Check answer: 0.68(fir) + 0.24(mm) + 0.62(qs) = 1.54
- Our work
    - Setup: 1.46 ms
    - Calculation: 0.07 ms = 2800 cycles
    - Check answer: 0.12 ms

- **Calculation improvement: 9300%**
- **Calculation cycle: 2800 cycles**

UART Result

- Original
    - interrupts to trigger ISR : 1
    - time spent in one ISR : 0.9 (ms) = 36000 (cycles)
- Our work
    - interrupts to trigger ISR : n
    - time spent in one ISR :

n	time	cycle	cycle / interrupt
2	0.48 (ms)	19200	9600
3	0.60 (ms)	24000	8000
4	0.72 (ms)	28800	7200
5	0.85 (ms)	34000	6800



Utilization

design_1_caravel_ps_0_0_utilization_synth.rpt

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	119	0	0	53200	0.22
LUT as Logic	119	0	0	53200	0.22
LUT as Memory	0	0	0	17400	0.00
Slice Registers	158	0	0	106400	0.15
Register as Flip Flop	158	0	0	106400	0.15
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

Ref Name	Used	Functional Category
FDRE	158	Flop & Latch
LUT3	79	LUT
LUT6	46	LUT
LUT2	8	LUT
LUT4	4	LUT
LUT5	1	LUT
LUT1	1	LUT

design_1_axi_uartlite_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	104	0	0	53200	0.20
LUT as Logic	86	0	0	53200	0.16
LUT as Memory	18	0	0	17400	0.10
LUT as Distributed RAM	0	0			
LUT as Shift Register	18	0			
Slice Registers	113	0	0	106400	0.11
Register as Flip Flop	113	0	0	106400	0.11
Register as Latch	0	0	0	106400	0.00
F7 Muxes	1	0	0	26600	<0.01
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	95	Flop & Latch
LUT6	29	LUT
LUT5	27	LUT
LUT4	27	LUT
SRL16E	18	Distributed Memory
FDSE	18	Flop & Latch
LUT2	14	LUT
LUT3	9	LUT
LUT1	2	LUT
MUXF7	1	MuxFx

design_1_blk_mem_gen_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	10	0	0	53200	0.02
LUT as Logic	8	0	0	53200	0.02
LUT as Memory	2	0	0	17400	0.01
LUT as Distributed RAM	0	0			
LUT as Shift Register	2	0			
Slice Registers	12	0	0	106400	0.01
Register as Flip Flop	12	0	0	106400	0.01
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	2	0	0	140	1.43
RAMB36/FIFO*	2	0	0	140	1.43
RAMB36E1 only	2				
RAMB18	0	0	0	280	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	12	Flop & Latch
LUT2	6	LUT
SRL16E	2	Distributed Memory
RAMB36E1	2	Block Memory
LUT4	2	LUT

design_1_output_pin_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	10	0	0	53200	0.02
LUT as Logic	10	0	0	53200	0.02
LUT as Memory	0	0	0	17400	0.00
Slice Registers	12	0	0	106400	0.01
Register as Flip Flop	12	0	0	106400	0.01
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	12	Flop & Latch
LUT5	4	LUT
LUT4	4	LUT
LUT6	1	LUT
LUT2	1	LUT
LUT1	1	LUT

design_1_caravel_0_0_utilization_synth.rpt

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	7052	0	0	53200	13.26
LUT as Logic	6774	0	0	53200	12.73
LUT as Memory	278	0	0	17400	1.60
LUT as Distributed RAM	240	0			
LUT as Shift Register	38	0			
Slice Registers	7608	0	0	106400	7.15
Register as Flip Flop	7608	0	0	106400	7.15
Register as Latch	0	0	0	106400	0.00
F7 Muxes	298	0	0	26600	1.12
F8 Muxes	79	0	0	13300	0.59

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	7	0	0	140	5.00
RAMB36/FIFO*	4	0	0	140	2.86
RAMB36E1 only	4				
RAMB18	6	0	0	280	2.14
RAMB18E1 only	6				

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	4	0	0	220	1.82
DSP48E1 only	4				

7. Primitives

Ref Name	Used	Functional Category
FDCE	4207	Flop & Latch
LUT6	3040	LUT
FDRE	3029	Flop & Latch
LUT4	1542	LUT
LUT5	1452	LUT
LUT2	627	LUT
LUT3	586	LUT
LUT1	424	LUT
CARRY4	419	CarryLogic
MUXF7	298	MuxFx
FDPE	283	Flop & Latch
RAMS32	232	Distributed Memory
FDSE	89	Flop & Latch
MUXF8	79	MuxFx
SRL16E	38	Distributed Memory
RAMD32	24	Distributed Memory
RAMB18E1	6	Block Memory
RAMB36E1	4	Block Memory
DSP48E1	4	Block Arithmetic

design_1_auto_us_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	192	0	0	53200	0.36
LUT as Logic	168	0	0	53200	0.32
LUT as Memory	24	0	0	17400	0.14
LUT as Distributed RAM	0	0			
LUT as Shift Register	24	0			
Slice Registers	343	0	0	106400	0.32
Register as Flip Flop	343	0	0	106400	0.32
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	341	Flop & Latch
LUT3	84	LUT
LUT6	81	LUT
LUT5	28	LUT
SRLC32E	24	Distributed Memory
LUT4	19	LUT
LUT2	12	LUT
LUT1	4	LUT
FDSE	2	Flop & Latch

design_1_rst_ps7_0_10M_0_utilization_synth.rp

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	19	0	0	53200	0.04
LUT as Logic	18	0	0	53200	0.03
LUT as Memory	1	0	0	17400	<0.01
LUT as Distributed RAM	0	0			
LUT as Shift Register	1	0			
Slice Registers	40	0	0	106400	0.04
Register as Flip Flop	40	0	0	106400	0.04
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	36	Flop & Latch
LUT2	9	LUT
LUT4	6	LUT
LUT1	5	LUT
FDSE	4	Flop & Latch
LUT5	3	LUT
SRL16E	1	Distributed Memory
LUT6	1	LUT
LUT3	1	LUT

design_1_read_romcode_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	739	0	0	53200	1.39
LUT as Logic	664	0	0	53200	1.25
LUT as Memory	75	0	0	17400	0.43
LUT as Distributed RAM	0	0			
LUT as Shift Register	75	0			
Slice Registers	1100	0	0	106400	1.03
Register as Flip Flop	1100	0	0	106400	1.03
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	1	0	0	140	0.71
RAMB36/FIFO*	1	0	0	140	0.71
RAMB36E1 only	1				
RAMB18	0	0	0	280	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	1097	Flop & Latch
LUT3	261	LUT
LUT6	212	LUT
LUT4	158	LUT
LUT2	132	LUT
SRL16E	75	Distributed Memory
LUT5	68	LUT
CARRY4	63	CarryLogic
LUT1	22	LUT
FDSE	3	Flop & Latch
RAMB36E1	1	Block Memory

design_1_spiflash_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	44	0	0	53200	0.08
LUT as Logic	44	0	0	53200	0.08
LUT as Memory	0	0	0	17400	0.00
Slice Registers	63	0	0	106400	0.06
Register as Flip Flop	63	0	0	106400	0.06
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	32	Flop & Latch
FDCE	31	Flop & Latch
LUT3	26	LUT
LUT6	21	LUT
CARRY4	10	CarryLogic
LUT4	5	LUT
LUT5	4	LUT
LUT1	2	LUT
LUT2	1	LUT

design_1_auto_pc_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	210	0	0	53200	0.39
LUT as Logic	208	0	0	53200	0.39
LUT as Memory	2	0	0	17400	0.01
LUT as Distributed RAM	2	0			
LUT as Shift Register	0	0			
Slice Registers	230	0	0	106400	0.22
Register as Flip Flop	230	0	0	106400	0.22
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	196	Flop & Latch
LUT5	131	LUT
LUT6	25	LUT
FDCE	23	Flop & Latch
LUT4	22	LUT
LUT3	21	LUT
LUT2	20	LUT
LUT1	16	LUT
CARRY4	16	CarryLogic
FDPE	11	Flop & Latch
RAMD32	2	Distributed Memory

design_1_auto_pc_1_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	421	0	0	53200	0.79
LUT as Logic	356	0	0	53200	0.67
LUT as Memory	65	0	0	17400	0.37
LUT as Distributed RAM	0	0			
LUT as Shift Register	65	0			
Slice Registers	562	0	0	106400	0.53
Register as Flip Flop	562	0	0	106400	0.53
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	546	Flop & Latch
LUT3	236	LUT
LUT6	130	LUT
LUT5	53	LUT
SRLC32E	47	Distributed Memory
LUT4	43	LUT
LUT2	19	LUT
SRL16E	18	Distributed Memory
CARRY4	18	CarryLogic
FDSE	16	Flop & Latch
LUT1	5	LUT

design_1_axi_intc_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	70	0	0	53200	0.13
LUT as Logic	70	0	0	53200	0.13
LUT as Memory	0	0	0	17400	0.00
Slice Registers	65	0	0	106400	0.06
Register as Flip Flop	65	0	0	106400	0.06
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	63	Flop & Latch
LUT3	29	LUT
LUT6	24	LUT
LUT5	12	LUT
LUT4	11	LUT
LUT2	11	LUT
LUT1	2	LUT
FDSE	2	Flop & Latch

design_1_processing_system7_0_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	24	0	0	53200	0.05
LUT as Logic	24	0	0	53200	0.05
LUT as Memory	0	0	0	17400	0.00
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	0	0	0	125	0.00
Bonded IPADs	0	0	0	2	0.00
Bonded IOPADs	130	0	0	130	100.00
PHY_CONTROL	0	0	0	4	0.00
PHASER_REF	0	0	0	4	0.00
OUT_FIFO	0	0	0	16	0.00
IN_FIFO	0	0	0	16	0.00
IDELAYCTRL	0	0	0	4	0.00
IBUFDS	0	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	200	0.00
ILOGIC	0	0	0	125	0.00
OLOGIC	0	0	0	125	0.00

5. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	1	0	0	32	3.13
BUFIO	0	0	0	16	0.00
MMCME2_ADV	0	0	0	4	0.00
PLLE2_ADV	0	0	0	4	0.00
BUFMRCE	0	0	0	8	0.00
BUFHCE	0	0	0	72	0.00
BUFR	0	0	0	16	0.00

7. Primitives

Ref Name	Used	Functional Category
BIBUF	130	IO
LUT1	24	LUT
PS7	1	Specialized Resource
BUFG	1	Clock

design_1_xbar_0_utilization_synth.rpt

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	220	0	0	53200	0.41
LUT as Logic	220	0	0	53200	0.41
LUT as Memory	0	0	0	17400	0.00
Slice Registers	134	0	0	106400	0.13
Register as Flip Flop	134	0	0	106400	0.13
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	134	Flop & Latch
LUT6	124	LUT
LUT4	62	LUT
LUT5	33	LUT
LUT3	13	LUT
LUT2	11	LUT
LUT1	1	LUT