1a) Consider the following spatial program, where a floating point addition takes 12 clock cycles:

```
SCSVar a=io.input(‘‘a’’, SCSFloat(20,6));
SCSVar b=io.input(‘‘b’’, SCSFloat(20,6));
SCSVar fb = newFeedback(SCSFloat(20,6));
SCSVar r = fb + a ∗ b;
fb = feedback(r);
io.output(‘‘r’’, r, SCSFloat(64, 12));
```

Assume the CPU sends arrays a,b to the spatial computing substrate and receives the partial multiply-accumulated sums back. Write the CPU pseudo code that takes the resulting array r[N] and prints the result of the vector-vector multiply. Explain your answer.

b) If we assume that a fixed point addition can be achieved in a single clock cycle, how do you convert the above code to fixed point? For example, if for a floating point variable, the minimal value is -100.00 and the maximal value is 1000.00, and the output needs to be accurate to within 0.1, what is the resulting fixed point type? How does the source code on the CPU side change?

The two parts carry equal marks.

2a) Given a spatial computing substrate with 10MB of SRAM, 14-by-14 multipliers and 96GB of attached DRAM, describe an efficient implementation (pseudo code) of the kernel for 28-bit (integer) iterative matrix multiplication v=A*v for an N-by-N 2D matrix A and vector v, for the case of N=3. Assume A[i] gives you access to the i-th element of A.

b) How does the implementation for part 1 above change for N=100, and N=10,000? Where do you store the data? Describe how you load the data in each case? How does the manager support the access of data in each case?

c) Given DRAM bandwidth of 100GB per second and a spatial computing substrate operating at 1GHz, how long would it take to calculate 10,000 iterations of the matrix-vector multiply?

The three parts carry, respectively, 40%, 40%, and 20% of the marks.

3a) Let us assume that we need to compute a complex function which can be approximated by a polynomial. We need our function f(x) to have a minimal maximum error of e and can get the optimal coefficients c[i] for the polynomial from a suitable mathematics package. On the spatial computing side, we could use any fixed point or floating point format with any number of bits. We can also decide how many coefficients we should use.

i) Show the code for a simple spatial kernel implementing a polynomial with 4 coefficients, streaming in x and streaming out the resulting function value. Assume 32 bit fixed point values with a binary dot in the middle. Minimize the space taken by the arithmetic units in your solution.

ii) Assuming that we want to minimize the size of the computational unit on the spatial computing substrate, devise a scheme to decide how many coefficients we should use and how many bits each of the fields (e.g. mantissa, exponent, etc) needs to have to maintain our minimal maximum error e.

b) List and describe six rounding modes.

ii) For the polynomial above, what would be the impact on space and precision in using different rounding modes? Why would you use “round to nearest” versus “round to nearest even”? What would be the most efficient rounding mode in space and precision?

iii) For a fixed point multiply of two numbers, each with eight integer bits and four fractional bits, fix(8,4), how many integer and fractional bits are in the result of the multiply? If we want to keep only the same number of bits as the inputs, we need to convert the result back to fix(8,4). How many bits do we need to look at to decide on “round to nearest even”? If the numbers get too large, we might get overflow. What

are our options to deal with overflow in a computer with decoupled control and dataflow computing units?

The two parts carry equal marks.