Solutions for Chapter 8

Exercise 8.1

a. Given the equation of the matching function

$$\sum_{i=1}^{N}\sum_{j=1}^{N}[x(i,j)-y(i,j)]^2,$$

the type and number of operations can directly be derived:
Per data pair, 1 subtraction and 1 multiplication with accumulation have to be performed.
The following table shows the required operations and their respective cycle count for processors 1 and 2:

operation	no./operations	cycles/proc.1	cycles/proc.2
SUB	1	1	1
MAC	1	16	1
matching func.	**2**	**17**	**2**

The ratio between the cycle counts of both processors is 17/2.

b. Again, the given matching function

$$\sum_{i=1}^{N}\sum_{j=1}^{N}|x(i,j)-y(i,j)|$$

directly leads to the required operations per data pair:
1 subtraction, 1 absolute value, and 1 addition.

operation	no./operations	cycles/proc.1	cycles/proc.2
SUB	1	1	1
ABS	1	1	1
ADD	1	1	1
matching func.	**3**	**3**	**3**

The ratio between the cycle counts of both processors is now 1.

c. According to (3.5.18), the rotation of a vector is given by:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

In a first step, the number of operations required for computation of sin and cos shall be determined. In (3.5.6), the polynomial approximation is given as:

$$\sin\theta \approx \theta \cdot P_{3,s}(\theta^2)$$

$$\theta \in [0, \pi/2]$$

$$\cos\theta \approx P_{3,c}(\theta^2).$$

After applying the Horner-scheme, the type and number of operations can be determined from the following formulae:

$$\sin\theta \approx \theta \cdot (((a_{s3} \cdot \theta^2 + a_{s2}) \cdot \theta^2 + a_{s1}) \cdot \theta^2 + a_{s0})$$

$$\cos\theta \approx ((a_{c3} \cdot \theta^2 + a_{c2}) \cdot \theta^2 + a_{c1}) \cdot \theta^2 + a_{c0}$$

In following, it is assumed that the term θ^2 was precomputed using a multiplication and stored in memory. Therefore, the computation of sinθ and cosθ requires:

sinθ: 4 MUL, 3 ADD.

cosθ: 3 MUL, 3 ADD.

Rotation of vector $[x_0, y_0]^T$ by θ is computed using the following equations:

$$x_n = x_0 \cdot \cos\theta - y_0 \cdot \sin\theta$$

$$y_n = x_0 \cdot \sin\theta + y_0 \cdot \cos\theta$$

This leads to:

4 MUL, 1 ADD, and 1 SUB.

In total, the vector rotation requires the following computations: once θ^2, once sinθ and cosθ and the equations for x_n and y_n. In total,

12 MUL, 7 ADD, and 1 SUB

are necessary to perform the vector rotation.

The total number of cycles for both processors is given in the following table:

operation	**no./operations**	**cycles/proc.1**	**cycles/proc.2**
MUL	12	192	12
SUB	7	7	7
SUB	1	1	1
vector rotation	**20**	**200**	**20**

d. The CORDIC iteration equations accoring to (3.5.28) are:

$$x_{i+1} = x_i - \sigma_i 2^{-i} y_i$$

$$y_{i+1} = y_i - \sigma_i 2^{-i} x_i \qquad \sigma_i \in \{-1, 1\}$$

$$z_{i+1} = z_i - \sigma_i \tan^{-1}(2^{-i})$$

For each of the 16 steps of the shift sequence the following steps have to be performed:

- Determination of σ_i by comparing z_i to 0.
  Required operation: SUB
- Calculation of x_{i+1} depending on the actual value of σ_i.
  This calulation requires one shift operation by *i* and one ADD resp. SUB operation depending on σ_i.
- Calculation of y_{i+1} depending on the actual value of σ_i.
  This calulation requires one shift operation by *i* and one ADD resp. SUB operation depending on σ_i.
- Calculation of z_{i+1} depending on the actual value of σ_i. As the values for $\tan^{-1}$ are stored in memory, only one ADD resp. SUB operation depending on σ_i is required.

Per shift step the following number of operations is required:

1 SUB, 3 ADD/SUB and 2 ASH by *i* bit.

This results in the total number of operations for the CORDIC method for 16 shift steps:

16 SUB, 48 ADD/SUB und 32 ASH by *i* Bit.

Depending on the number of bits to shift, both processors require a different number of clock cycles. The following table lists the cycles per shift position for *i*=0...15 and the sum:

i	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***Σ***
cycl./ proc.1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	***120***
cycl./ proc.2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	***15***

Finally, there are 2 multiplications necessary to scale the amplitude. This leads to the total amount of cycles per processor:

	SUB	**SUB/ADD**	**ASH**	**MUL**	**Total**
cycl./proc.1	16	48	240	32	336
cycl./proc.2	16	48	30	2	96

Exercise 8.2

a. For a matrix-vector multiplication, 1 multiplication and N–1 multiply-accumulate operations are required. As the number of cycles for MUL and MAC is for each processor the same, the number of operations can be simplified to N multiply-accumlate operations.

Per NxN block, $2N$ matrix-vector multiplications have to be performed. Thus, $2N^2$ MAC operations are required. For N=8, this leads to 128 operations.

The matching function requires N^2 subtractions, N^2 absolut values and N^2–1 additions per block.
This leads for $(1+2N)^2$ search points to:

$$(1 + 2N)^2 \cdot N^2 \qquad \text{subtractions}$$

$$(1 + 2N)^2 \cdot N^2 \qquad \text{absolute values}$$

$$(1 + 2N)^2 \cdot (N^2 - 1) \qquad \text{additions}$$

For N=8 the number of operations is: 18,496 SUB, 18,496 ABS, and 18,207 ADD.

In total, both algorithms require the following number of operation per input data and the respective number of cycles per processor. :

	MAC	**ADD**	**SUB**	**ABS**	**Total**
operations per input data	2	284.5	289	289	864.5
cycles proc. 1	32	284.5	289	289	894.5
cycles proc. 2	2	284.5	289	289	864.5

b. According to the first row of the table in a.), the number of operations per input data is 864,5. This results in a throughput of 57,837 input data at 50MHz clock rate.

c. For processor 2, the throughput of 57,837 input data at 50MHz clock rate was correctly determined in b.).
Because the number of cycles for the MUL operation of processor 1 is not equal to 1, the number of cycles required for the algorithm can’t be determined using the number of operations. Thus, for processor 1, the data from the second row of the table has to be taken for the computation of the througput rate. This leads to a throughput of 55,897 input data per second.

d. The new search method reduces the amount of operations to:

$(1 + 8 \log N) \cdot N^2$ subtractions

$(1 + 8 \log N) \cdot N^2$ absolut values

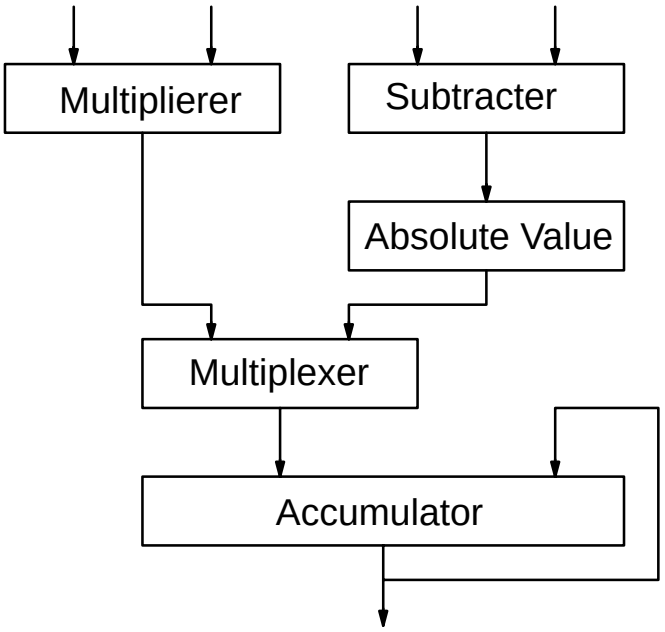
$(1 + 8 \log N) \cdot (N^2 - 1)$ additions

per data block. With N=8, 1,600 subtractions, 1,600 absolut values, and 1,575 additions have to be performed. Additionally, 128 multiply-accululate operations per block for the calculation of the matrix-vector multiplication are required.
Per input data, this leads to the following table:

	MAC	**ADD**	**SUB**	**ABS**	**Total**
operations per input data	2	24.6	25	25	76.6
cycles proc. 1	32	24.6	25	25	106.6
cycles proc. 2	2	24.6	25	25	76.6

Now, the throughput rate is increased to 652,742 input data processed at 50 MHz clock rate.

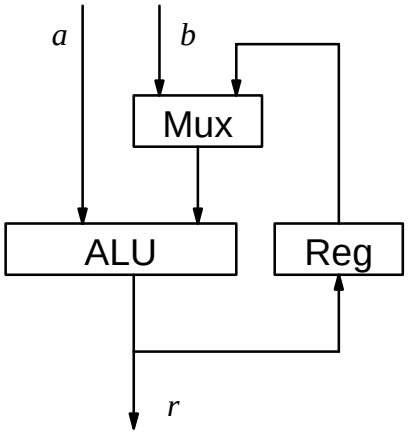
e. The data path shown below allows for the computation of the matching function in one clock cycle if pipelining is applied. As before, the matrix-vector multiplication requires 1 cycle for the multiply-accumulation operation. In total, 2 cycles are required for the matrix-vector multiplication for each data of the matrix.



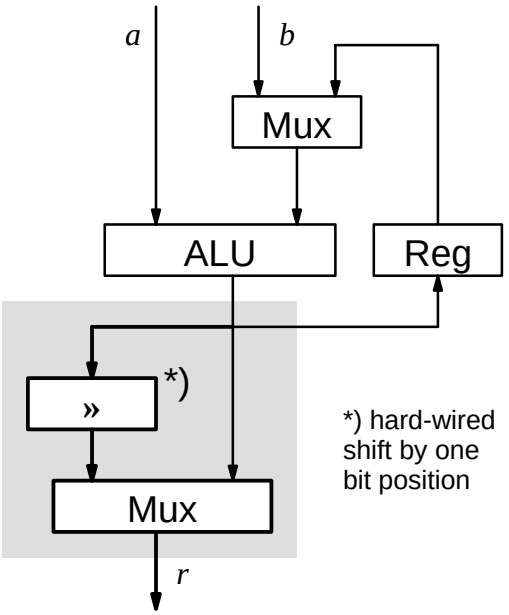
Taking N=8 and $(1+2N)^2$ = 289 search points, 289 clock cycles per input data for the matching function and 2 clock cycles per input data for the matrix-vector multiplication are required. This leads to a throughput of 171,821 input data per second at 50 MHz.

For N=8 and 1+8 log N = 25 search points, 27 clock cycles per input data are required. This leads to a throughput of 1,851,852 input data per second at 50 MHz.

Exercise 8.3



Data path (basis version)



Data path for average operation

a. Average of two operands:

The average of two operands can almost completely be performed by the ALU itself. In $r = (a + b)/2$, the addition of a and b is done by the ALU. The division by 2 can

be implemented using a right shift by 1 position. This can be realized by feeding the output bus of the ALU shifted by one bit position to an additional multiplexer input.

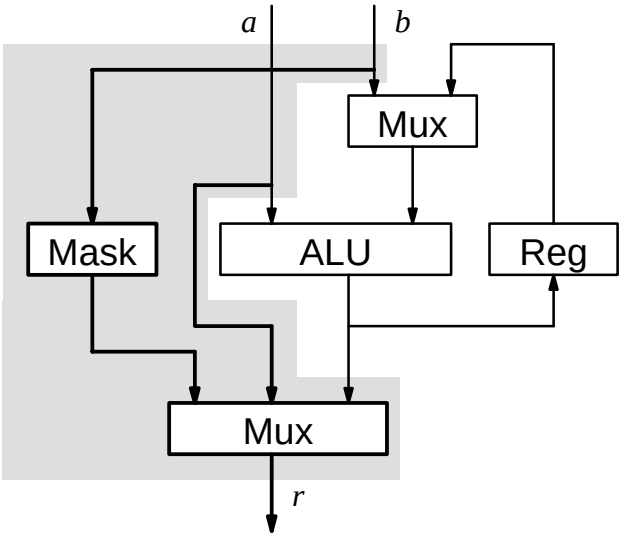
b. Saturation:

The saturation is performed using a compare operation of *a* and *N* and depending on the result either *a* or *N*–1 is selected using an output multiplexer. The compare operation is done by the ALU using a CMP or SUB operation. The value of *N*–1 is generated by a seperate mask operation.

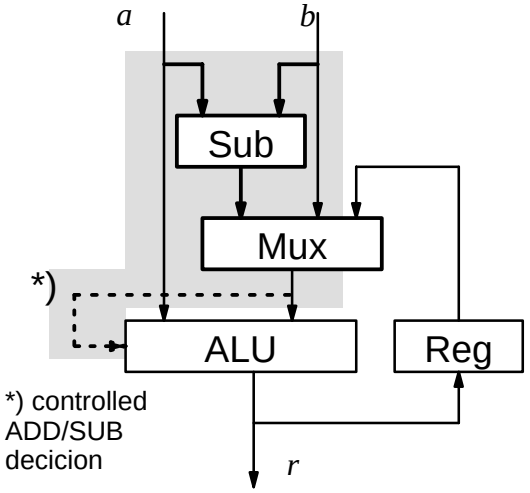
The mask operation is realized by a bitslice with inputs x_{i+1} and m_{i+1} and output m_i, where $m_i = x_{i+1} \vee m_{i+1}$ and the most significant output bit being zero.

c. Controlled ADD/SUB

This case can easily be handled by using the sign of *a* as control input to the ALU. For $sign(a) = 0$ the ALU is set in ADD mode, for $sign(a) = 1$ the ALU is set in SUB mode.



Saturate to N–1



Summation of magn. of differences

d. Sum of magnitude of differences

The difference is computed using a separate subtractor connected to a third input of the multiplexer. Accumulation and absolut value computation are performed using a controlled ADD/SUB operation of the ALU. In case the difference *a*–*b* is positive, the accumulation is done by an addition. If *a*–*b* is negative, a subtraction is used to perform the absolut value and accumulation in a single step.