```
(define (und-gatter a1 a2 ausgabe)  ;;; Und-Gatter
  (local
    ((define (und-vorgang-prozedur)
       (local ((define neuer-wert
                       (logisches-und (get-signal a1)
                                      (get-signal a2))))
        (verzoegert und-gatter-verzoegerung
           (lambda () (set-signal! ausgabe neuer-wert))))))

    (begin (add-vorgang! a1 und-vorgang-prozedur)
           (add-vorgang! a2 und-vorgang-prozedur))))

(define (logisches-und s1 s2)
  (if (and (= s1 1) (= s2 1))
      1
      0))
```



```
(define (inverter eingabe ausgabe)   ;;; Inverter
  (local
     ((define (invert-eingabe)
         (verzoegert inverter-verzoegerung
            (lambda ()
              (set-signal! ausgabe
                    (logisches-nicht (get-signal eingabe)))))))
    (add-vorgang! eingabe invert-eingabe)))

(define (logisches-nicht s)
  (cond ((= s 0) 1)
        ((= s 1) 0)
        (else (error 'not "Ungueltiges Signal"))))
```



```
(define (konstr-draht)     % Implementierung eines Drahts
  (local ((define signal-wert 0)
          (define vorgang-prozeduren empty)
    (define (set-mein-signal! neuer-wert)
      (if (not (= signal-wert neuer-wert))
          (begin (set! signal-wert neuer-wert)
                 (jede-aufrufen vorgang-prozeduren))
          'fertig))
    (define (add-vorgang-prozedur proz)
      (begin (set! vorgang-prozeduren
                   (cons proz vorgang-prozeduren))
             (proz)))
    (define (zuteilen m)
      (cond ((equal? m 'get-signal) signal-wert)
            ((equal? m 'set-signal!) set-mein-signal!)
            ((equal? m 'add-vorgang!) add-vorgang-prozedur)
            (else (error 'draht "Unbekannte Operation")))))
   zuteilen))
```



```
(define (jede-aufrufen prozeduren)
  (if (empty? prozeduren)
      'fertig
      (begin ((first prozeduren))
             (jede-aufrufen (rest prozeduren)))))

(define (get-signal draht)
  (draht 'get-signal))

(define (set-signal! draht neuer-wert)
  ((draht 'set-signal!) neuer-wert))

(define (add-vorgang! draht vorgang-prozedur)
  ((draht 'add-vorgang!) vorgang-prozedur))
```

```
;;; Realisierung der Verzoegerung

(define (verzoegert verzoegerung vorgang)
  (hinzufuegen-plan! (+ verzoegerung (aktuelle-zeit der-plan))
                     vorgang
                     der-plan))

;;; Oberste Ebene der Simulation

(define (fortfuehren)
  (if (leerer-plan? der-plan)
      'fertig
      (begin ((erster-plan-eintrag der-plan))
             (entferne-ersten-plan-eintrag! der-plan)
             (fortfuehren))))
```



```
;;; Struktur fuer Zeitsegmente:
(define-struct segment (zeit warteschlange))

;;; Struktur fuer Zeitplaene:
(define-struct plan (zeit segmente))

(define (konstr-plan) (make-plan 0 empty))

(define (aktuelle-zeit plan) (plan-zeit plan))

(define (set-aktuelle-zeit! plan zeit)
  (set-plan-zeit! plan zeit))

(define (segmente plan) (plan-segmente plan))

(define (set-segmente! plan segs)
  (set-plan-segmente! plan segs))

(define (erstes-segment plan) (mfirst (segmente plan)))

(define (rest-segmente plan) (mrest (segmente plan)))

(define (leerer-plan? plan) (empty? (segmente plan)))
```



```
(define (hinzufuegen-plan! zeit vorgang plan)
 (local
   ((define (gehoert-vor? segmente)
      (or (empty? segmente)
          (< zeit (segment-zeit (mfirst segmente)))))


    (define (hinzufuegen-segmente! segmente)
      (cond
        ((= (segment-zeit (mfirst segmente)) zeit)
          (hinzufuegen-warteschlange!
                 (segment-warteschlange (mfirst segmente))
                 vorgang))
        ((gehoert-vor? (mrest segmente))
          (set-mrest! segmente
                (mcons (konstr-neues-zeit-segment zeit vorgang)
                       (mrest segmente))))
        (else (hinzufuegen-segmente! (mrest segmente))))))
```



```
  (define (konstr-neues-zeit-segment zeit vorgang)
    (local ((define q (konstr-warteschlange)))
      (begin (hinzufuegen-warteschlange! q vorgang)
             (make-segment zeit q))))


;;; Rumpf von hinzufuegen-plan!:
(if (gehoert-vor? (segmente plan))
    (set-segmente! plan
           (mcons (konstr-neues-zeit-segment zeit vorgang)
                  (segmente plan)))
    (hinzufuegen-segmente! (segmente plan))))
```

```
(define (entferne-ersten-plan-eintrag! plan)
  (local
    ((define q (segment-warteschlange (erstes-segment plan))))
    (begin (entfernen-warteschlange! q)
           (if (leere-warteschlange? q)
               (set-segmente! plan (rest-segmente plan))
               'fertig))))

(define (erster-plan-eintrag plan)
  (if (leerer-plan? plan)
      (error 'erster-plan-eintrag "Plan ist leer")
      (local ((define seg1 (erstes-segment plan)))
        (begin (set-aktuelle-zeit! plan (segment-zeit seg1))
               (anfang (segment-warteschlange seg1))))))
```



```
;;; Anbringen von Sonden am Draht
(define (sonde name draht)
  (add-vorgang! draht
         (lambda () (begin (display name)
                           (display (aktuelle-zeit der-plan))
                           (display "  neuer-wert = ")
                           (display (get-signal draht))
                           (newline)))))

;;; Initialisierung der Simulation
(define der-plan (konstr-plan))

(define inverter-verzoegerung 2)
(define und-gatter-verzoegerung 3)
(define oder-gatter-verzoegerung 5)
```



```
;;; Definiere Drähte:
(define a1 (konstr-draht))
(define a2 (konstr-draht))
(define s  (konstr-draht))
(define c  (konstr-draht))

;;; Anbringen von Sonden:
(sonde "Draht a1: " a1)
(sonde "Draht a2: " a2)
(sonde "Draht s: "  s)
(sonde "Draht c: "  c)

;;; Definiere Schaltkreis (Gatter):
(halbaddierer a1 a2 s c)

;;; Setzen von Signalen und Ablauf der Simulation:
(set-signal! a1 1)
(fortfuehren)
(set-signal! a2 1)
(fortfuehren)
```