Opdracht 1

|  |
| --- |
| void statemachine (void)  {  /\* statemachine control \*/  static eSystem\_State currentState = INIT; // state-variable  static bit stateEntry = TRUE; // true upon entering a new state  eSystem\_State nextState = currentState;  /\* declare local user-variables below here \*/  /\* statemachine \*/  switch (currentState) {  case INIT:  if (stateEntry) {  // place ENTRY actions of STATE1 here  init\_statemachine();  }  else {  // STATE actions; change state if needed  nextState = LEDS\_OFF;  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  }  break;  case LEDS\_OFF:  if (stateEntry) {  // place ENTRY actions of STATE1 here  if(interval != 0)  P0 = 0;  }  else {  // STATE actions; change state if needed  nextState = LED0\_ON;  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  }  break;  case LED0\_ON:  if (stateEntry) {  // place ENTRY actions of STATE1 here  P0 |= 1 << 0;  }  else {  // STATE actions; change state if needed  nextState = LED1\_ON;  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  P0 &= ~(1<<0);  }  break;  case LED1\_ON:  if (stateEntry) {  // place ENTRY actions of STATE1 here  P0 |= 1 << 1;  }  else {  // STATE actions; change state if needed  nextState = LED0\_ON;  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  P0 &= ~(1<<1);  }  break;  default:  // internal error: not possible  currentState = LEDS\_OFF;  break;  }  /\* update FSM internals \*/  if (currentState != nextState)  {  /\* change of state \*/  stateEntry = TRUE;  currentState = nextState;  }  else if (stateEntry)  {  /\* state entered \*/  stateEntry = FALSE;  }  }  void init\_statemachine(void)  {  P0=0;  } |

Opdracht 2

|  |
| --- |
| #include "my\_statemachine.h"  /\* define your FSM states below here \*/  enum eSystem\_State{INIT, LEDS\_OFF, LED0\_ON, LED1\_ON};  typedef enum eSystem\_State eSystem\_State;  enum switchState{INITB, NOBTN, STOP, B1P, B2P, B21, B12, SNEL, LANGZAAM, ERROR};  typedef enum switchState switchState;  /\* define your (PRIVATE) function prototypes below here \*/  void updateContraints();  /\* Globaal omdat boeiend \*/  static char codering = 0;  static short interval = 10;  /\* FSM entry \*/  void statemachine (void)  {  /\* statemachine control \*/  static eSystem\_State currentState = INIT; // state-variable  static bit stateEntry = TRUE; // true upon entering a new state  eSystem\_State nextState = currentState;  /\* declare local user-variables below here \*/  static short intervalCount = 0;  /\* statemachine \*/  switch (currentState) {  case INIT:  if (stateEntry) {  // place ENTRY actions of STATE1 here  init\_statemachine();  }  else {  // STATE actions; change state if needed  nextState = LEDS\_OFF;  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  }  break;  case LEDS\_OFF:  if (stateEntry) {  // place ENTRY actions of STATE1 here  if(interval != 0)  P0 |= 1;  P0 |= 2;  }  else {  if(codering > 0)  updateContraints();  if(interval > 0)  nextState = LED0\_ON;  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  P0 &= ~(1<<0);  P0 &= ~(1<<1);  }  break;  case LED0\_ON:  if (stateEntry) {  // place ENTRY actions of STATE1 here  P0 |= 1 << 0;  }  else {  if(codering > 0)  updateContraints();  if(interval == 0)  nextState = LEDS\_OFF;  else if(intervalCount < interval)  intervalCount += 10;  else if(intervalCount >= interval)  {  intervalCount = 10;  nextState = LED1\_ON;  }  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  P0 &= ~(1<<0);  }  break;  case LED1\_ON:  if (stateEntry) {  // place ENTRY actions of STATE1 here  P0 |= 1 << 1;  }  else {  if(codering > 0)  updateContraints();  if(interval == 0)  nextState = LEDS\_OFF;  else if(intervalCount < interval)  intervalCount += 10;  else if(intervalCount >= interval)  {  intervalCount = 10;  nextState = LED0\_ON;  }  }  if (currentState != nextState) {  // place EXIT actions of STATE1 here  P0 &= ~(1<<1);  }  break;  default:  // internal error: not possible  currentState = LEDS\_OFF;  break;  }  /\* update FSM internals \*/  if (currentState != nextState)  {  /\* change of state \*/  stateEntry = TRUE;  currentState = nextState;  }  else if (stateEntry)  {  /\* state entered \*/  stateEntry = FALSE;  }  }  void init\_statemachine(void)  {  P0=0;  P0 |= 1<<7;  }  void switchStateMachine(void)  {  /\* statemachine control \*/  static switchState currentState = NOBTN; // state-variable  static bit stateEntry = TRUE; // true upon entering a new state  switchState nextState = currentState;  /\* declare local user-variables below here \*/  bit switch1 = SW1\_PRESSED();  bit switch2 = SW2\_PRESSED();  /\* statemachine \*/  switch (currentState) {  case INITB:  nextState = NOBTN;  break;  case NOBTN:  {  if(switch1 & switch2)  {  nextState = STOP;  }  else if(switch1)  {  nextState = B1P;  }  else if(switch2)  {  nextState = B2P;  }  }  break;  case STOP:  if (stateEntry) {  // place ENTRY actions of STATE1 here  codering = 1;  P0 |= 1<<6;  P0 |= 1<<7;  }  else {  // STATE actions; change state if needed  if(!switch1 & !switch2)  nextState = NOBTN;  }  if(currentState != nextState)  {  P0 &= ~(1<<6);  P0 &= ~(1<<7);  }  break;  case B1P:  if(stateEntry)  {  }  else  {  if(switch1 & switch2)  {  nextState = B12;  }  else if(!switch1 & !switch2)  {  nextState = SNEL;  }  }  break;  case B2P:  if(stateEntry)  {  }  else  {  if(switch1 && switch2)  {  nextState = B21;  }  else if(!switch1 & !switch2)  {  nextState = LANGZAAM;  }  }  break;  case B12:  if(stateEntry)  {  P0 |= 1<<6;  P0 &= ~(1<<7);  }  else  {  if(!switch1 & !switch2)  nextState = NOBTN;  }  break;  case B21:  if(stateEntry)  {  P0 |= 1<<7;  P0 &= ~(1<<6);  }  else  {  if(!switch1 & !switch2)  nextState = NOBTN;  }  break;  case SNEL:  if(stateEntry)  {  codering = 2;  updateContraints();  }  else  nextState = NOBTN;  break;  case LANGZAAM:  if(stateEntry)  {  codering = 3;  }  else  nextState = NOBTN;  break;  case ERROR:  if(stateEntry)  {  P0 |= 1<<2;  }  else  {  if(!switch1 & !switch2)  nextState = NOBTN;  }  if(nextState != currentState)  {  P0 &= ~(1<<2);  }  break;  default:  nextState = NOBTN;  break;  }  /\* update FSM internals \*/  if (currentState != nextState)  {  /\* change of state \*/  stateEntry = TRUE;  currentState = nextState;  }  else if (stateEntry)  {  /\* state entered \*/  stateEntry = FALSE;  }  }  void updateContraints(void)  {  //Als codering 1 is gaat de machine stopper  if(codering == 1)  interval = 0;  //als codering 2 is gaat de machine sneller  if(codering == 2)  {  if(interval == 0)  interval = 10;  if(interval == 10)  interval = 50;  else if(interval == 50)  interval = 250;  else if(interval == 250)  interval = 500;  else if(interval == 500)  interval = 1000;  else if(interval == 1000)  interval = 0;  }  //Als codering 3 is gaat de machine langzamer  if(codering == 3)  {  if(interval == 0)  interval = 1000;  else if(interval == 1000)  interval = 500;  else if(interval == 500)  interval = 250;  else if(interval == 250)  interval = 50;  else if(interval == 50)  interval = 10;  else if(interval == 10)  interval = 0;  }  codering = 0;  } |