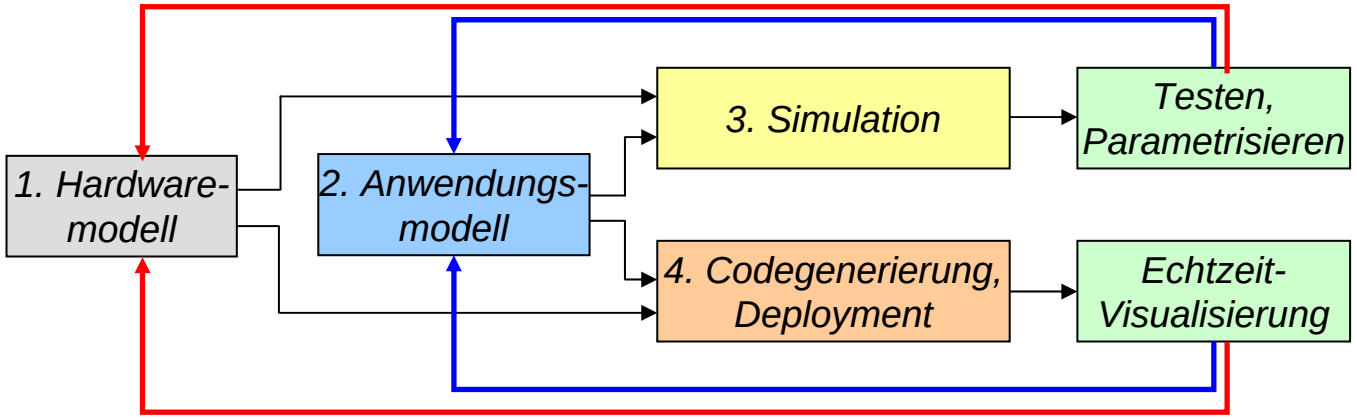
Modellierung von Echtzeitsystemen

Synchroner Datenfluss

Werkzeug: EasyLab

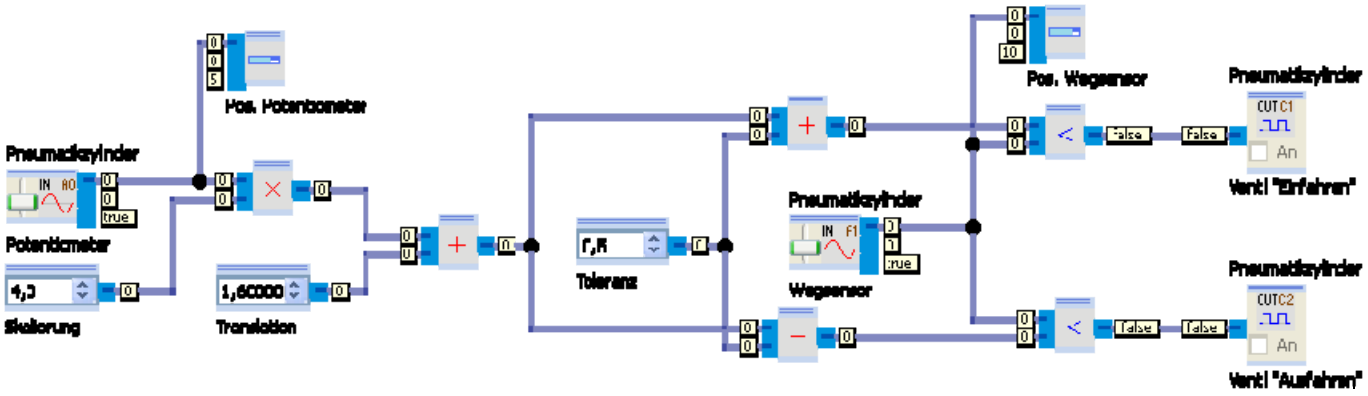
Entwicklungsprozess in EasyLab

1. Spezifikation der Zielhardware
2. Modellierung der Zustandslogik sowie der abzuarbeitenden Aufgabe je Zustand
3. Simulation des Programms zum Testen und zur Erkennung von Fehlern
4. Codegenerierung und Echtzeit-Visualisierung des Zustands der Zielhardware



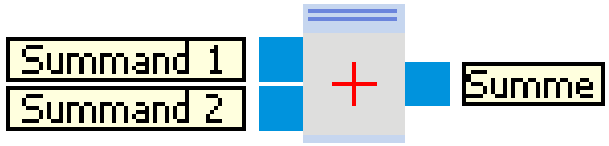
Synchroner Datenfluss

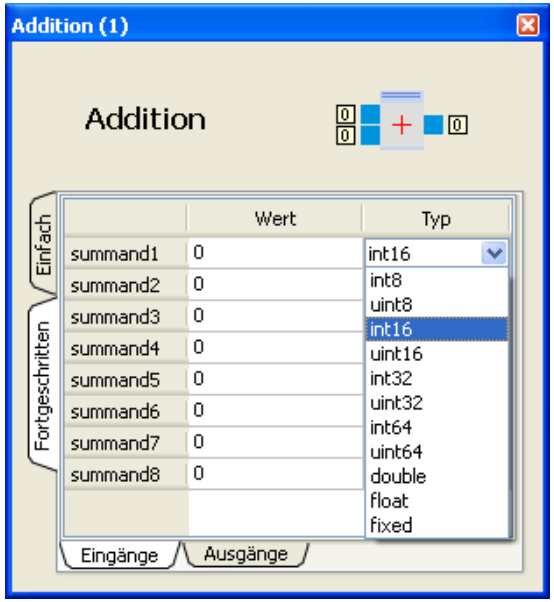
- Funktionsblöcke
  - Eingänge und Ausgänge
  - Interner Zustand
  - Zugeordnete Aktionen
- Komposition von Funktionsblöcken
  - Typisierte Verbindungen
  - In Bearbeitung: Hierarchische Funktionsblöcke
- Grundlagen
  - Synchronitätshypothese
  - „Black boxes“-Sicht
  - Effizienz und Zuverlässigkeit
    - Berechnung statischer Schedules
    - Deterministisches Laufzeitverhalten
    - Statische Speicherallokation (RAM)
    - Erzeugung kompakten Codes (ROM)



Überladen von Aktoren

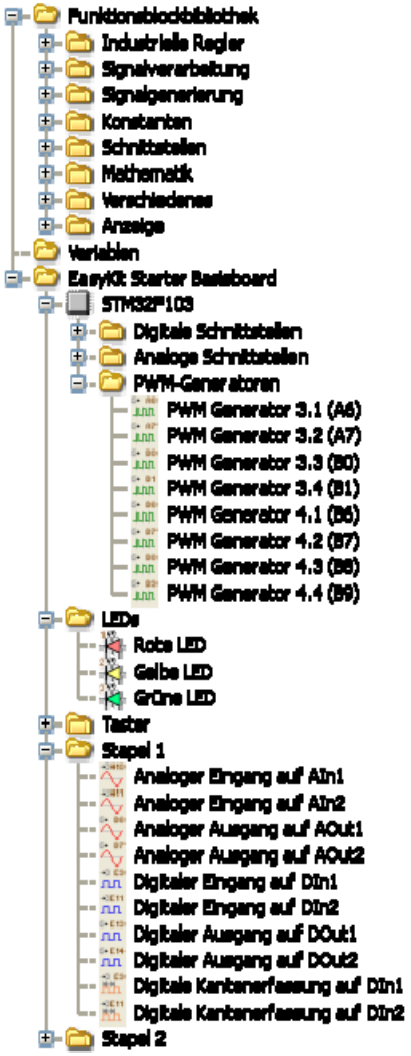
- Aktoren können typunabhängig angeboten werden
- Der Typ eines Aktors ist solange frei wählbar, bis sich durch Anschlussbelegung der Typ automatisch ergibt





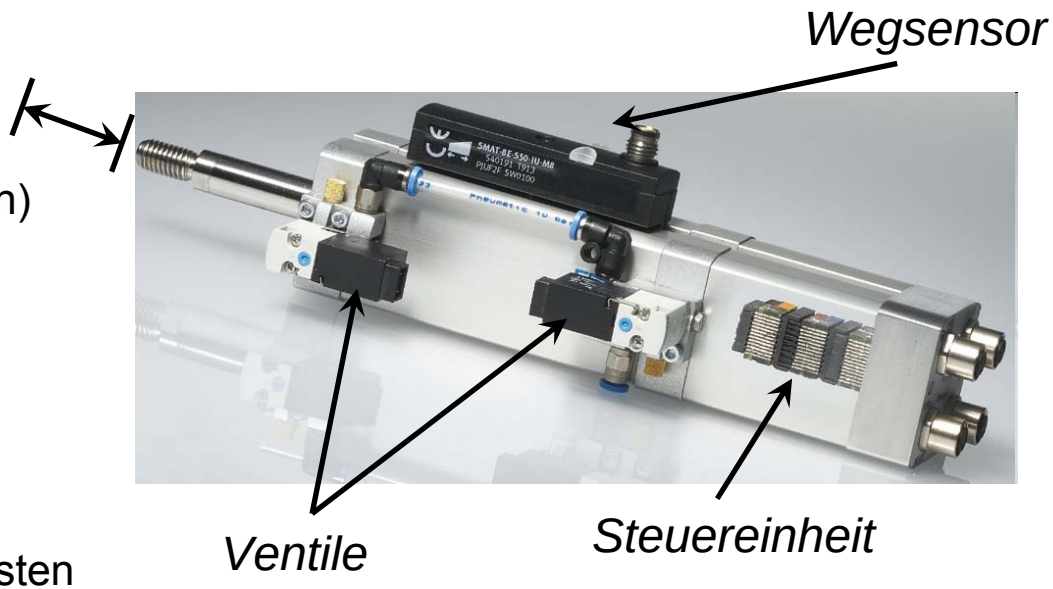
Geräte

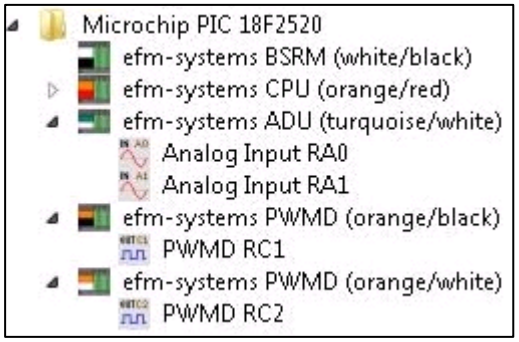
- Hardware
  - Mikrocontroller
  - Sensoren und Aktoren
- Geräte
  - Hierarchische Beschreibung der Hardware
  - Ressourcenmanagement
    - Modellierung der Hardwarefunktionalität
    - Beispiele: I/O-Pins, ADCs, Timer,
  - Schnittstelle zur Anwendungslogik
    - Hardwarezugriff
    - Geräte bieten eine Menge von Funktionsblöcken an



Anwendung – Pneumatischer Zylinder

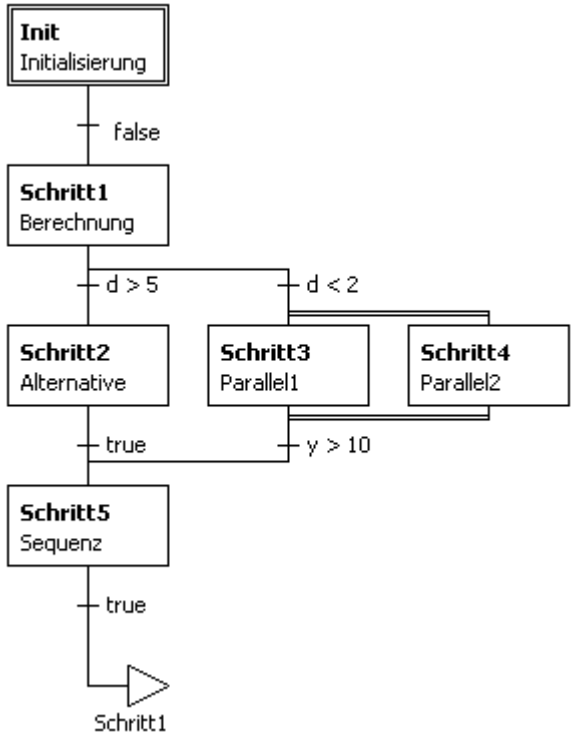
- Hardware
  - Zylinder
    - Positionssensor (Kolben)
    - Endlagenschalter
    - Zwei Magnetventile
  - Steuerungseinheit
    - Mikrocontroller
    - Analog-Digital-Wandler
    - Treiber für induktive Lasten
- Ziel: Positionssteuerung des Kolbens
- Umsetzung
  - Hardware-Modell aus Bibliothek für Match-X
  - Anwendungsmodell
    - Kleines Datenflussdiagramm
    - Integration der Hardwarefunktionalität





EasyLab: Zustandsfluss-Diagramme

- Zustandsfluss-Diagramme
  - An IEC-61131-3 angelehnt
  - Zustand: Referenz auf SDF-Modell
  - Transitionsbedingungen:
    Boolesche Ausdrücke mit Variablen
- Komposition von Zuständen
  - Zustandsfolgen
  - Alternativ- und Parallelzweige
  - Sprünge
- Vorteile
  - Modellierung von Zuständen
    (vgl. Automaten)
  - Diagrammart weit verbreitet in
    Anwendungsdomäne



Perspektiven

- Ausdehnung der Modellierung auf **verteilte Systeme**
- Automatische Parallelisierung für **Multi-** und **Many-Core-Systeme:** Besseren Verhältnisses von Leistung / Energieeinsatz
- Berücksichtigung nicht-funktionaler Eigenschaften
  - Automatische Erzeugung energieeffizienten Codes
  - Fehlertoleranz-Mechanismen
  - Quality of Service
- Verstärker Einsatz in **Lehre und Ausbildung**
  - Mikrocontroller-Praktikum
  - Übung zur Vorlesung Echtzeitsysteme

Modellierung von Echtzeitsystemen

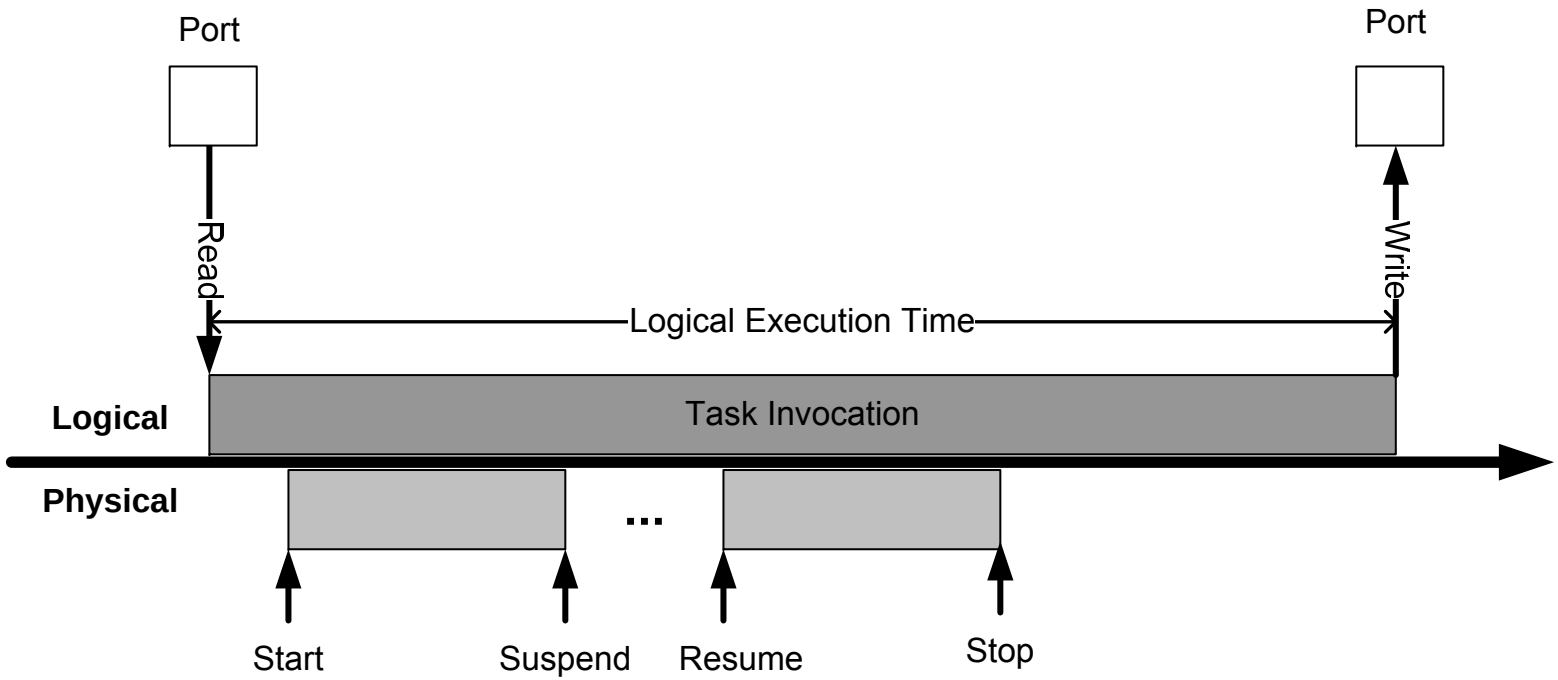
Zeitgesteuerte Systeme

Werkzeug: Giotto

Giotto: Hintergrund

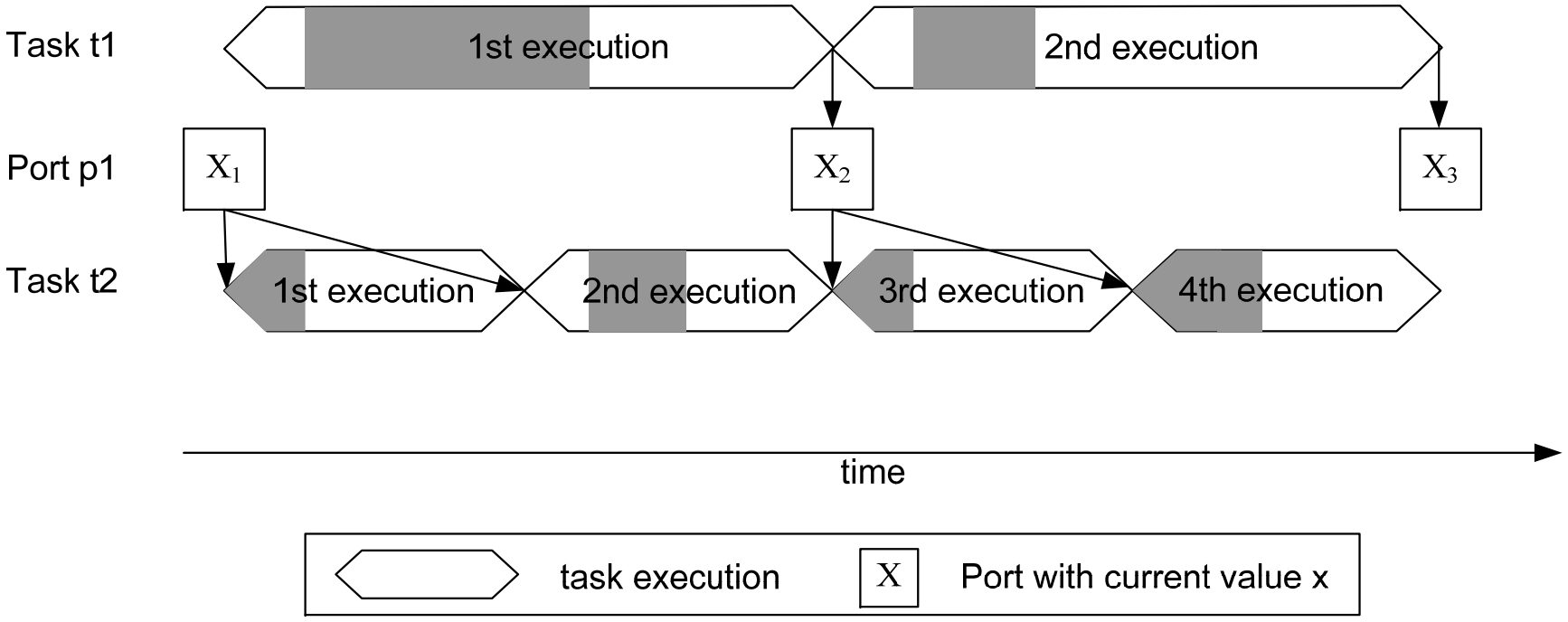
- Programmierumgebung für eingebettete Systeme (evtl. ausgeführt im verteilten System)
- Ziel:
  - strikte Trennung von plattformunabhängiger Funktionalität und plattformabhängigen Scheduling und Kommunikation
  - temporaler Determinismus
- Hauptkonzept: Logische Ausführungszeiten
- Aktoren:
  - Tasks
    - Programmblock aus sequentiellen Code
    - keine Synchronisationspunkte, blockende Operationen erlaubt
    - Schnittstellen: Ports
  - Drivers: realisieren die Kommunikation zwischen Ports
  - Flexibilität durch Modes/Guards
- Ausführung durch virtuelle Maschinen:
  - Embedded Machine: Reaktion der Tasks auf physikalische Ereignisse
  - Scheduling Machine: physikalisches Scheduling
- http://embedded.eecs.berkeley.edu/giotto/

Logische Ausführungszeit



Motivation siehe http://www.cs.uic.edu/~shatz/SEES/henzinger.slides.ppt

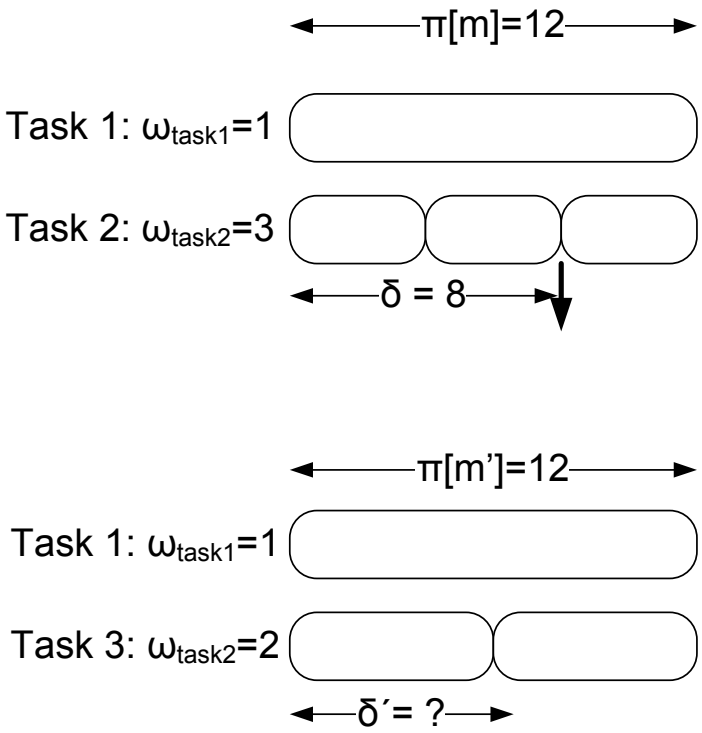
Kommunikation zwischen Tasks



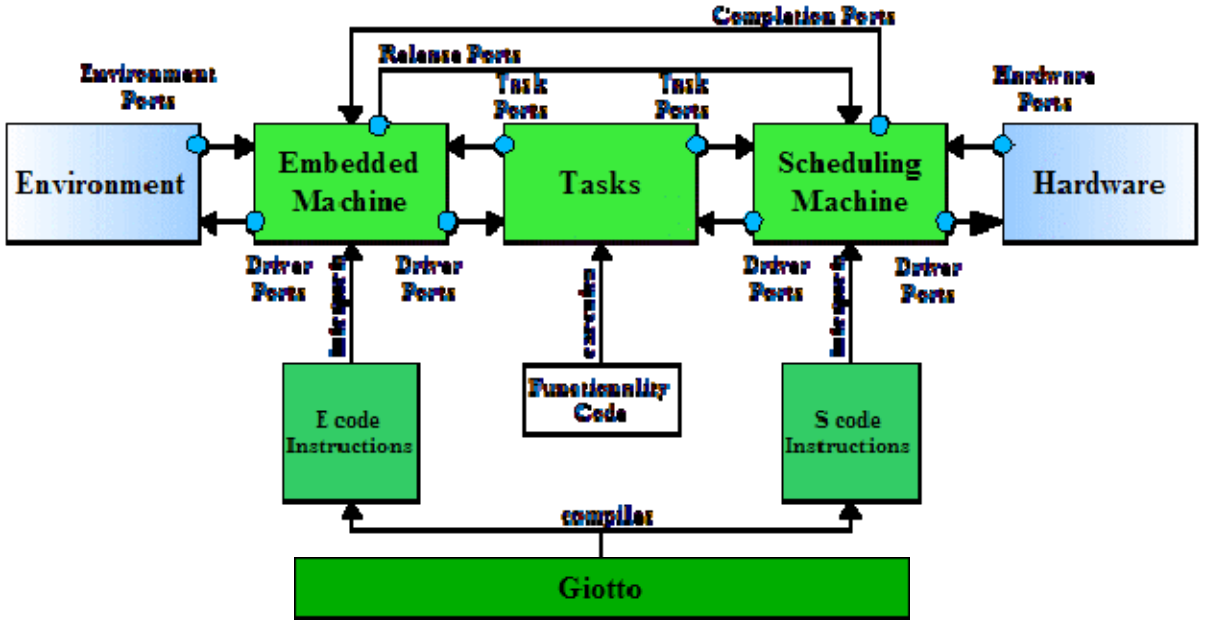
Modes / Guards

- Um die Ausführung flexibel zu gestalten, bietet Giotto Modes und Guards an
  - Guards: Boolsche Funktion, die über die Ausführung eines Tasks entscheidet (wird vor Start des Tasks aufgerufen)
  - Mode: Menge von Tasks und Drivers die zeitgleich ausgeführ werden, es kann immer nur ein Mode aktiv sein.
- Nicht-harmonischer Moduswechsel (Unterbrechung eines laufenden Modes):
  - Voraussetzung: $\pi[m]/\omega_{task} = \pi[m']/\omega'_{task}$
    m: Quellmodus, m': Zielmodus, $\pi[m]$: Modusdauer m, ω_{task}: Taskfrequenz $\Rightarrow$ Logische Ausführungszeit muss gleich sein
  - Wechselmechanismus:

    γ = LCM $\{\pi[m]/\omega_{task} | \{\omega_{task}, t,) \in$ Invokes[m],
    $\delta' = \pi[m']-(\varepsilon-\delta)$ mit $\varepsilon = n^* \gamma \geq \delta$
    LCM: least common multiple, δ: aktuelle Rundenzeit, δ': neue Rundenzeit in m', $\varepsilon - \delta$: Zeit bis zum nächsten gleichzeitigen Beendigungspunkt



Ausführungsumgebung



Zusammenfassung

- Das Konzept der logischen Ausführungszeiten erlaubt eine Abstrahierung von der physikalischen Ausführungszeit und somit die Trennung von plattformunabhängigem Verhalten (Funktionalität und zeitl. Verhalten) und plattformabhängiger Realisierung (Scheduling, Kommunikation)
- Die Ausführung erfolgt über zwei virtuelle Maschinen:
  - E-Machine: Interaktion mit der Umgebung (reaktiv)
  - S-Machine: Interaktion mit der ausführenden Plattform (proaktiv), Vorteil: Schedule kann vorab berechnet werden
- Weitere Literaturhinweise:
  - Henzinger et al.: Giotto: A time-triggered language für embedded programming, Proceedings of the IEEE, vol.91, no.1, pp. 84-99, Jan 2003
  - Henzinger et al.: Schedule-Carrying Code, Proceedings of the Third International Conference on Embedded Software (EMSOFT), 2003