Regelungsentwicklung mit MATLAB/Simulink in Verbindung mit SiL und HiL auf Basis von Open-Source-Hardware

M.Eng. Andre Klingenschmidt, B.Sc. Robin Peuker
RESOL - Elektronische Regelungen GmbH
Heiskampstr. 10, 45527 Hattingen
Tel.: 02324 9648 0, Mail: andre.klingenschmidt@resol.de, robin.peuker@resol.de

Abstract

Der Beitrag liefert die grundsätzlichen Ansätze für die Implementierung von Software- sowie Hardware-in-the-Loop-Schnittstellen auf Open-Source-Hardware in Verbindung mit MATLAB/Simulink. Dabei werden sowohl die konkrete programmiertechnische Umsetzung als auch die zu beachtenden Randbedingungen diskutiert. Anwendungshinweise und -szenarien werden vorgestellt und um konkrete Beispiele und Ergebnisse aus der industriellen Praxis ergänzt. Techniken wie SiL und HiL sind keine realitätsferne Theorie, vielmehr sind sie in der Lage den Entwicklungsbetrieb zu professionalisieren, was sich aufgrund der erzielbaren Zeitersparnis insbesondere auch für kleinere Unternehmen lohnt.

Software in the Loop

Software in the Loop (SiL) beschreibt im Rahmen des Rapid-Control-Prototyping (RCP) einen Aufbau, in welchem die Regelungssoftware auf dem Entwicklungsrechner anstatt auf der Zielhardware ausgeführt, jedoch trotzdem der reale Prozess geregelt wird. Dadurch werden Entwicklungs- sowie Debugzeit verkürzt und gleichzeitig die Qualität erhöht, da die Entwicklungsumgebung mehr Möglichkeiten für Diagnose und Optimierung bereithält. Um die bidirektionale Kommunikation, die für den Austausch von Sensor- und Aktorinformationen zwischen Entwicklungsrechner und Prozess nötig ist, herzustellen, ist eine Hardwareschnittstelle erforderlich [1]. Open-Source-Hardware wie Arduino, RaspberryPi o. ä. eignet sich durch die einfache Programmierbarkeit, hohe Flexibilität, Verfügbarkeit zahlreicher Erweiterungsplatinen (Shields) und geringe Beschaffungskosten sehr gut für diese Aufgabe. Die SiL-Schnittstelle muss zu Folgendem in der Lage sein:

- Erfassen der Messwerte der Anlagensensoren
- Beeinflussung/Ansteuerung der Anlagenaktoren
- Kommunikation mit dem Entwicklungsrechner
- Abarbeitung in ausreichend kurzer Zykluszeit

Abbildung 1: RESOLino Pt1000-Shield

Listing 1: TCP-Server in C++ auf Arduino

```
#include <Ethernet2.h>

int Port = 2222;
byte mac[] = {0x00,0x00,0x00,0x00,0x00,0x00};

EthernetServer tcpServer(Port);

void setup() {
  Serial.begin(115200);

  Ethernet.begin(mac);
  tcpServer.begin();

  Serial.println(Ethernet.localIP());
}
```

Die soeben aufgeführten Punkte werden folgend entsprechend am Beispiel eines Arduino Unos als SiL-Schnittstelle und MATLAB/Simulink als Entwicklungsumgebung besprochen. Sensorsignale wie Spannungen, Frequenzen und Impulse können oft direkt oder mit einfachen Vorschaltungen (Pull-Up/Down, Spannungsteiler etc.) mit einem Arduino gemessen werden. Stromsignale oder Widerstandstemperatursensoren erfordern etwas umfangreichere Schaltungen. Hierfür sind jedoch bereits fertige Shields verfügbar, die einen direkten Anschluss dieser Signalarten ermöglichen. So z. B. das RESOLino Pt1000-Shield in Abbildung 1, welches zur Messung elektrischer Widerstände geeignet ist. Für die Ausgabe von Aktorwerten stehen auf dem Arduino digitale Ports zur Verfügung mit denen Relais geschaltet und Frequenz- oder PWM-Signale erzeugt werden können. Serielle Ein- und Ausgänge lassen sich bei Implementierung der nötigen Kommunikationsprotokolle durch die Verfügbarkeit von Soft- und Hardware-Serial ebenfalls flexibel einbinden. So lässt sich beispielhaft auf einem Arduino Uno mit aufgesetztem Ethernet-Shield durch den in Systeme mit eigenem Betriebssystem, wie z. B. ein RaspberryPi, besitzen keine Echtzeitfähigkeit, können also nicht garantieren, dass Prozesse immer gleich lang dauern oder Ereignisse zu einem bestimmten Zeitpunkt auftreten. Reine Mikroprozessoren ohne Betriebssystem sind bei entsprechender Programmierung echtzeitfähig. Es muss im Einzelfall entschieden werden, inwiefern die Echtzeitfähigkeit relevant ist. Die im Hause RESOL bearbeiteten Systeme weisen oft schnelle Systemdynamiken auf und erfordern entsprechend kurze Zykluszeiten bei Messung und Regelung. Um diese zu garantieren werden als Hardwareschnittstellen in der Regel Arduinos verwendet.

Listing 2: TCP-Client in Level-2 MATLAB S-Function

```
global clientSiL;
if isempty(clientSiL)
  clientSiL = tcpclient(IP, 2222);
end
```

Listing 3: SiL-Kommunikation Client-Seite in Level-2 MATLAB S-Function

```
if answerReceived
    cmd = sprintf('SET,%d,%d,%d,%d',...
                  u(1), u(2), u(3), u(4));
    write(clientSiL, uint8(cmd));
    answerReceived = 0;
end

Bytes = 0;
timeout = 0; tic;
while Bytes == 0 && timeout < 0.015
    Bytes = get(clientSiL, 'BytesAvailable');
    timeout = toc;
end

if Bytes > 0
    answer = char(read(clientSiL));
    answer_parts = strsplit(answer, ',');
    answerReceived = 1;
end
```

Listing 1 gezeigten kurzen Codeabschnitt ein TCP-Server erzeugen. Wird die IP-Adresse über DHCP zugeteilt, müssen lediglich MAC-Adresse und Portnummer definiert werden. Die erhaltene IP-Adresse kann über den seriellen Monitor ausgegeben werden. In der Entwicklungsumgebung ist nun die Implementierung der Gegenstelle erforderlich. Bei Verwendung von Simulink eignet sich hierfür eine Level-2 MATLAB S-Function. Seit der Version 2014b beinhaltet MATLAB standardmäßig sowohl TCP-Client als auch -Server. Es ist also keine zusätzliche Toolbox notwendig. Listing 2 zeigt die Erzeugung des TCP-Clients mit bekannter Portnummer und IP-Adresse. Der Client wird global angelegt, um in allen Subfunktionen Zugriff zu gewähren. So kann beispielsweise bei Aufruf des Terminate-Callbacks noch ein letzter Steuerbefehl gesendet werden, der die Anlage in einen sicheren Zustand versetzt.
In Listing 3 ist der Kommunikationsablauf auf Client-Seite (Simulink) innerhalb des Output-Callbacks der S-Function dargestellt. Die persistente Variable *answerReceived* blockiert das Senden eines neuen Stellbefehls, bis die Antwort auf den letzten Stellbefehl eingetroffen ist. Die Antwort enthält die aktuellen Sensorwerte. Während eines Callback-Aufrufs wird maximal 15 ms auf eine Antwort gewartet, um eine zu lange Blockierung des Programmablaufs zu vermeiden.

Listing 4: SiL-Kommunikation Server-Seite in C++ auf Arduino

```
1  int byteRead = 0;
2  char buffer[256];
3  EthernetClient tcpClient;
4
5  // Stellbefehl lesen
6  tcpClient = tcpServer.available();
7  if (tcpClient) {
8    clientConnected = true;
9    byteRead = tcpClient.available();
10   tcpClient.readBytes(buffer, byteRead);
11 }
12 // Zerlegung von buffer mit strtok()
13 ...
14 // Messwerte senden
15 printf(buffer, "DATA,%d,%d,%d",
16        data[0], data[1], data[2]);
17 tcpClient.print(buffer);
```

Auf dem Arduino werden permanent die folgenden Schritte im loop-Teil des Programms abgearbeitet:

- Einlesen der Messdaten
- Einlesen der Daten von einem TCP-Client, wenn verfügbar
- Senden der aktuellen Messdaten an den TCP-Client
- Setzen der Aktorwerte

Wenn ein TCP-Client verfügbar ist, so werden die von ihm gesendeten Daten wie in Listing 4 dargestellt in einen Puffer gelesen. Dieses char-Array kann dann ebenfalls an einem definierten Trennzeichen zerlegt und die Bestandteile (i. e. die Aktorwerte) verarbeitet werden. Die Zerlegung des char-Arrays ist in C/C++ nicht so elegant möglich wie in MATLAB, mit *strtok(buffer, ",")* steht aber auch hier eine gut dokumentierte Variante zur Verfügung. Anschließend werden als Antwort auf den Stellbefehl die aktuellen Messwerte versendet.
Abbildung 2 zeigt das SiL-Blockschaltbild in Simulink. Die SiL-S-Function nimmt die Aktorwerte der Regelung entgegen, sendet sie an die SiL-Schnittstelle, bekommt als Antwort die aktuellen Sensorwerte vom Prozess und gibt diese zur Verarbeitung in der Regelung und zur Diagnose zurück. Um zum einen algebraische Schleifen zu vermeiden und zum anderen möglichst dieselbe Abtastrate wie auf der Zielhardware zu erhalten, sind den Ausgängen der SiL-S-Function Rate-Transition- und Zero-Order-Hold-Blöcke mit der gewünschten Samplerate nachgeschaltet.

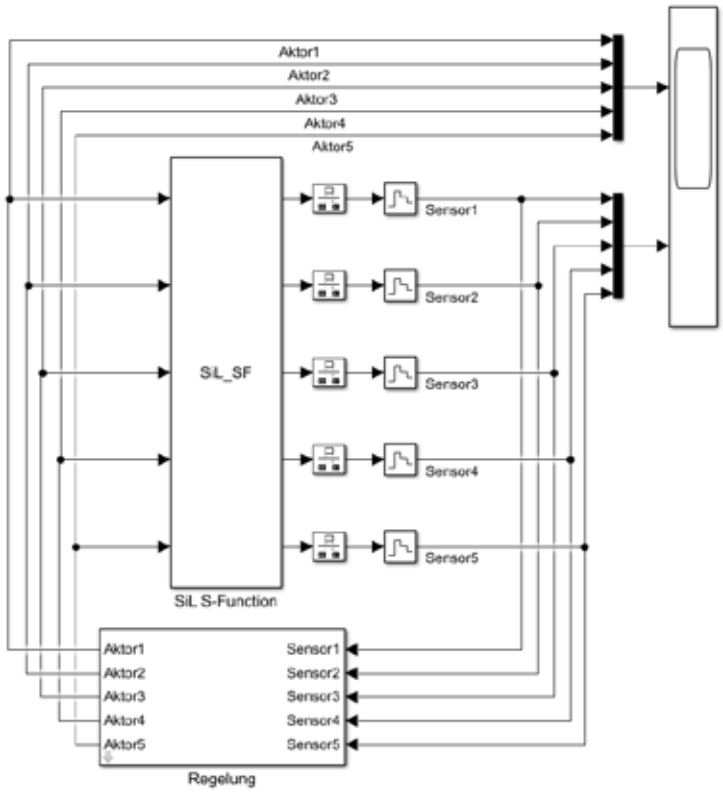

Abbildung 2: SiL-Blockschaltbild in Simulink

Die Sampletimes bzw. Zykluszeiten des SiL-Konstrukts sind eine Kernangelegenheit. Im Normalfall sollten die Abtast- respektive Abfrageraten der Sensorik und der Aufruftakt des Regelalgorithmus den Zeiten entsprechen, die später auf der Zielhardware erreicht werden. Eine Diskrepanz an dieser Stelle kann eine Fehlerquelle bei der Portierung von der Entwicklungs- zur Zielhardware darstellen. Zur Untersuchung von Vor- und Nachteilen kürzerer oder längerer Abtastzeiten ist eine Abweichung jedoch zulässig.
Die in Simulink einstellbare Berechnungsschrittweite hat keinen Bezug zur real ablaufenden Zeit. Die Blöcke müssen jedoch in den festgelegten Zeitabständen aufgerufen werden. Dazu muss die Berechnungsschrittweite kleiner oder gleich den angestrebten Zykluszeiten (i. e. Block-Sampletimes) gewählt werden. Um die absolute Korrektheit der Zeitstempel zu gewährleisten, muss dafür gesorgt werden, dass die Simulationszeit gleich der Realzeit ist. Sollte die Simulationszeit langsamer voranschreiten als die Realzeit, wird das in der Regel durch zu häufig aktualisierte Scopes oder tatsächlich durch eine zu geringe Performance des Entwicklungsrechners verursacht. Ersteres lässt sich durch geringere Scope-Samplerates oder durch Schließen der Scopes während der Simulation beheben.

Optimalerweise läuft die Simulationszeit also etwas schneller als die Realzeit, was letztlich ein Indikator für die Performancereserve des Entwicklungssystems ist. Um hier nun die Gleichheit zwischen Simulations- und Realzeit herzustellen, kann in der SiL-S-Function eine Realzeitbremse implementiert werden. Wie in Listing 5 gezeigt, wird zunächst die Differenz zwischen der aktuellen Simulationszeit und der tatsächlich vergangenen Zeit seit Simulationsstart ermittelt. Ist die Simulationszeit bereits weiter vorangeschritten, so wird mittels einer Java-Funktion die Simulation für die Dauer der Differenz angehalten. Die Java-Funktion ist ohne Weiteres direkt zugänglich und eignet sich an dieser Stelle besser als die MATLAB-Funktion *pause()*, da Letztere eine schlechtere und zudem systemabhängige Genauigkeit aufweist, insbesondere bei kurzen Pausenzeiten von wenigen Millisekunden [2]. Ungeachtet der beschriebenen Maßnahmen kann die Korrektheit der Zeitstempel nicht garantiert werden, da das Betriebssystem im Normalfall nicht echtzeitfähig ist und somit Schwankungen nicht vollständig beseitigt werden können. Um hier eine höhere Genauigkeit zu erreichen, kann alternativ die Realzeit ebenfalls als Vektor zusammen mit den Datenreihen gespeichert und im Anschluss an die Simulation zur Darstellung und Auswertung verwendet werden.

Listing 5: Realzeitbremse in Level-2 MATLAB S-Function

```
1  //Start-Callback
2  global startTime;
3  startTime = tic;
4  ...
5  //Output-Callback
6  global startTime;
7  realTdiff = (block.CurrentTime - toc(startTime));
8  if realTdiff > 0
9      java.lang.Thread.sleep(realTdiff * 1000);
10 end
```

Hardware in the Loop

Hardware in the Loop (HiL) beschreibt im Rahmen des RCP die Anbindung der Zielhardware an eine Prozesssimulation statt an die reale Anlage. Um dies zu erreichen, muss ähnlich wie bei SiL eine Schnittstelle geschaffen werden, welche die von der auf dem Entwicklungsrechner ausgeführten Simulation produzierten Sensorwerte so emuliert, dass sie über die vorgesehenen realen Sensoreingänge der Zielhardware erfasst werden können. Zeitgleich müssen die von der Zielhardware erzeugten Aktorsignale von der Schnittstelle aufgenommen und der Simulation zugeführt werden. Mit diesem Konstrukt sind Validierungen der Portierung von der Entwicklungs- zur Zielhardware sowie reproduzierbare Testszenarien und Regressionstests durchführbar [1]. Anstatt einer Live-Simulation können auch aufgezeichnete Messwerte als Datenquelle dienen. Die HiL-Schnittstelle muss zu folgendem in der Lage sein:

- Erfassen der Aktorwerte der Testhardware
- Emulieren der Anlagensensoren mit Werten aus der Simulation
- Kommunikation mit dem Entwicklungsrechner
- Abarbeitung in ausreichend kurzer Zykluszeit

Die HiL-Schnittstelle kann als Spiegelbild der SiL-Schnittstelle verstanden werden. Während bei SiL die Sensorwerte der realen Anlage gemessen werden müssen, werden diese bei HiL emuliert. Entsprechendes gilt

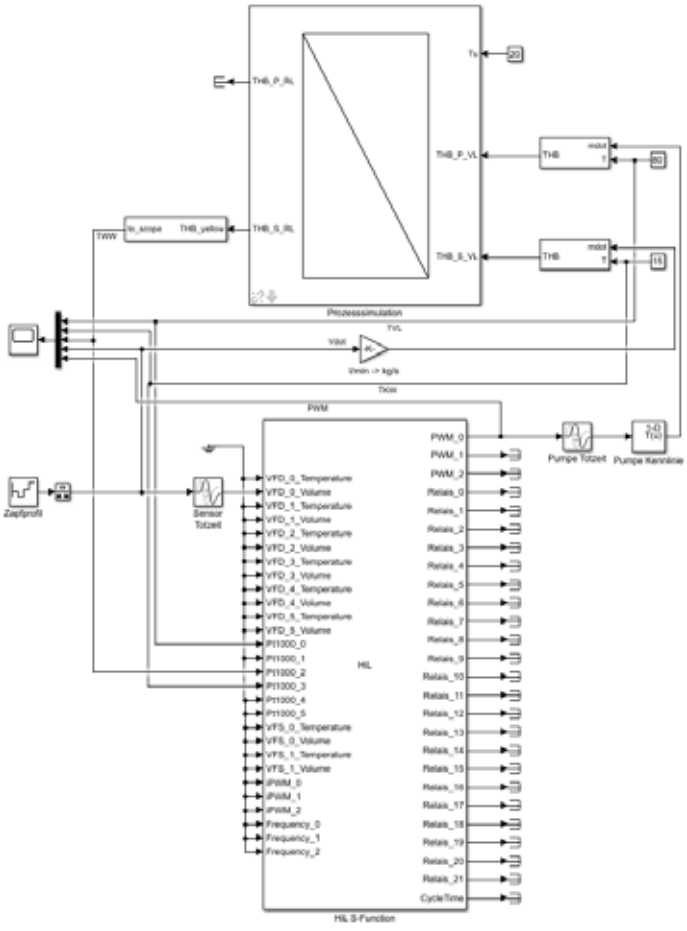

Abbildung 3: HiL-Blockschaltbild in Simulink

für die Aktorseite. Als Schnittstellenhardware eignen sich besonders Arduinos oder ähnliche Mikrocontroller. RaspberryPi und andere Einplatinenrechner mit Betriebssystem sind aufgrund des fehlenden Echtzeitbetriebssystems weniger gut geeignet.

Je nach Sensortyp können etwas unübliche Schaltungen zur Emulation nötig sein. Ein Widerstandstemperatursensor ließe sich beispielsweise über eine ansteuerbare Widerstandskaskade emulieren. Viele Sensoren arbeiten jedoch mit Frequenz- und Spannungssignalen oder über serielle Schnittstellen, welche sich wiederum auch auf einem Arduino unkompliziert implementieren lassen. Bei digitalen Sensoren oder Aktoren, die ein zeitkritisches Protokoll verwenden, kann es sinnvoll sein, Zusatzplatinen zu verwenden (ebenfalls Arduino o. ä.), die seriell an die Schnittstelle angebunden sind und diese Aufgaben erledigen. Die Geschwindigkeit von Sensoremulation, Aktorerfassung und der Kommunikation mit dem Entwicklungsrechner muss auf die Arbeitsgeschwindigkeit (Mess- und Regelungszyklus) der Testhardware abgestimmt sein. Das Nyquist-Theorem ist folglich zu be-

achten. Die Kommunikationsimplementierung kann genau so erfolgen wie bei SiL. Die HiL-S-Function in MATLAB versendet nun allerdings die Sensorwerte aus der laufenden Simulation an die HiL-Schnittstelle und nimmt als Antwort die von der Testhardware erzeugten Aktorwerte entgegen.
Wichtig ist die Zeitabstimmung zwischen der Prozesssimulation in MATLAB/Simulink und der Zielhardware. Die einfachste Lösung ist auch hier das Ausbremsen der Simulation auf Realzeit mit der in Listing 5 dargestellten Funktion. Denkbar wäre jedoch auch die Synchronisation beider Seiten auf eine andere Zeitbasis. Das kann nützlich sein, wenn ein Test schneller ablaufen soll als die Realzeit oder falls die Simulation langsamer abläuft als die Realzeit. Eine solche Abstimmung und Synchronisation müsste allerdings an mehreren Stellen gleichzeitig stattfinden und gestaltet sich daher deutlich schwieriger. In jedem Fall ist bei der Bewertung der Synchronisation darauf zu achten, dass die gesamte Prozesskette beachtet wird, also beispielsweise auch die Dauer der Sensoremulation. Abbildung 3 zeigt eine beispielhafte Implementierung in Simulink. Die Aktorwerte aus der Testhardware werden von der HiL-S-Function in das Prozessmodell geführt. Das Modell wird berechnet und gibt die Sensorwerte an die HiL-S-Function zurück. Aktor- und Sensoreigenschaften wie Totzeit, Trägheit oder Sättigung sollten ebenfalls modelliert werden. Die Auswahl des ODE-Solvers und der Berechnungsschrittweite müssen sich primär nach den Modellanforderungen richten.
Es existieren unterschiedliche Anwendungsarten für einen HiL-Test, die in den verschiedenen Entwicklungsstadien Verwendung finden:

- **Manuelle Bedienung**: Durch manuelle Einstellung der zu emulierenden Sensorwerte können beliebige Szenarien erzeugt werden. Diese Form der Bedienung ist im frühen Entwicklungsstadium zur Funktionsprüfung der Zielhardware oder auch später für Schnelltests sinnvoll. Rückkopplungsprozesse sollten auf diese Art nicht getestet werden.
- **Daten aus Simulation**: Wie bei SiL wird die HiL-Schnittstelle in eine Prozesssimulation, beispielsweise in MATLAB/Simulink eingebunden. Dafür muss natürlich ein entsprechendes Prozessmodell vorhanden sein. Diese Art der HiL-Nutzung eignet sich, um die fertiggestellte Zielhardware mit der vom Entwicklungsrechner portierten Regelungssoftware zu testen.
- **Daten aus Aufzeichnung**: Anstatt einer Simulation können auch aufgezeichnete Messdaten als Quelle für die zu emulierenden Sensorwerte dienen. Die Messdaten können von einem SiL-Test, aus dem Labor oder von einer Feldanlage stammen. Die Aktorwerte der Testhardware können hier die Sensorwerte nicht mehr beeinflussen. Es kann jedoch ein Abgleich der erzeugten Aktorwerte mit den Aktorwerte aus den Messdaten erfolgen. Hiermit sind reproduzierbare Tests möglich, die beispielsweise bei Fehlersuche oder Regressionstests neuer Softwareversionen nützlich sind. Je nach Zeitauflösung der Messdaten ist evtl. eine Interpolation notwendig.

Anwendung

Die Anwendungsbereiche der vorgestellten Techniken gestalten sich äußerst vielfältig. An erster Stelle steht natürlich das Prototyping und die allgemeine Regelungsentwicklung, welche die konsequente Anwendung von SiL und HiL deutlich beschleunigt und in ihrer Qualität und Robustheit verbessert wird. Algorithmusänderungen können sofort am realen Prozess erprobt und Messdaten zur Modellerstellung einfach gesammelt werden. Die Diagnose von fehlerhaftem Verhalten ist mit maximalem Informationsgewinn möglich.

Das Verhalten der Zielhardware kann gegen das Verhalten der Entwicklungshardware geprüft und Abweichungen schnell aufgedeckt werden. Die gewonnene Zeit kann in Produktverbesserungen, Neuentwicklungen sowie in die Erhöhung der Testdichte investiert werden. Mit SiL und HiL können geloggte Feldmessdaten, die einen Fehlerfall repräsentieren, direkt verwendet werden, um diesen innerhalb der Entwicklungsumgebung zu reproduzieren. Fehleridentifikation und -lösung werden beschleunigt und professionalisiert. Insbesondere eignet sich HiL für Release-Tests. Hier können reale Messdaten, die einst zu einem Fehler geführt haben automatisiert jeder neuen Softwareversion vorgesetzt werden. Damit kann gewährleistet werden, dass erfolgte Änderungen keine Seiteneffekte verursachen, die bereits gelöste Probleme wiederherstellen.

Außerhalb des Entwicklungsbetriebs dient vornehmlich SiL dem schnellen Testdesign zur Klärung offener Fragen aus den Bereichen der Vorentwicklung oder aus Innovationsprozessen. Beispielhaft seien hier die Qualifizierung neuer Sensorik, die Reaktionszeitmessung neuer Aktorik oder Machbarkeitsprüfungen auf Basis von Kundenanfragen oder neuer Produktideen genannt. Mit einer flexiblen SiL-Implementierung kann hier einfach und schnell ein Testaufbau erfolgen. Die Auswertung und Bearbeitung der gewonnenen Daten mithilfe von Tools wie MATLAB erlauben belastbare und wiederholbare Aussagen. Datenfilterung und -aufbereitung sowie die Untersuchung mit statistischen Methoden, Korrelationsprüfungen etc. sind unkompliziert möglich.

Ebenso ist die Fernanwendung möglich, so wurde die Anlage eines RESOL-Kunden, welche aufgrund ihrer Größe nicht im hauseigenen Labor vermessen werden konnte, im Labor des Kunden betrieben und war dabei mittels der SiL-Schnittstelle über das Internet mit dem Entwicklungsrechner bei RESOL verbunden.

Die Regelungsentwicklung für das Forschungsprojekt SH-T-Opt in Zusammenarbeit mit dem ISFH und Helma wurde ebenfalls mit der Unterstützung von SiL umgesetzt. Die Anlage im Experimentalhaus in Hannover wurde während der Entwicklung zeitweise direkt vom Entwicklungsrechner in Hattingen gesteuert [3].

Neben der Nutzung der Open-Source-Hardware als SiL- oder HiL-Schnittstelle kann diese weiterführend zur Erhöhung des Automatisierungsgrads in Testlaboren eingesetzt werden. Hier eignen sich Systeme mit Betriebssystem wie RaspberryPi oder BeagleBone besonders gut, da Steuerungen mit Webinterfaces zur Visualisierung sehr einfach implementiert werden können. Die Flexibilität ist hier deutlich höher als beim Einsatz einer SPS bei gleichzeitig geringerer Investition. Die Vorzüge einer SPS (Robustheit, sehr hohe Betriebssicherheit etc.) sind in diesen Einsatzszenarien meist nicht erforderlich.

Als Ergänzung folgen einige allgemeine Hinweise zum Umgang mit Open-Source-Systemen:

- **Screw-Shields**: Um den Anschluss von Leitungen an der Hardware zu vereinfachen, empfiehlt sich die Nutzung von Screw-Shields. Diese besitzen robuste Schraubklemmen anstatt empfindlicher Steckbuchsen. Zudem bieten sie oft die Möglichkeit, zusätzliche Masseklemmen (GND) anzubringen, da in der Regel jedes angeschlossene Element auch eine Masseverbindung benötigt, die Boards aus Platzgründen jedoch meist nur wenige Massebuchsen besitzen.
- **Erdung**: Um den Einfluss von EMV-Störungen und kleineren Potenzialdifferenzen zu eliminieren, hilft es meist, die Masseklemme des Entwicklungsboards zu erden.

- **Spannungsversorgung**: Netzteile mit einer schlechten Qualität der erzeugten Spannung können für verschiedene Probleme sorgen, die gelegentlich mit Softwarefehlern verwechselt werden und so zu langer und ergebnisloser Fehlersuche führen können. Es sollten entsprechend hochwertige Steckernetzteile verwendet werden. Bei Boards wie dem RaspberryPi sollte zudem auf eine ausreichend hohe Stromstärke von mind. 2 A geachtet werden.
- **Recherche**: Durch die hohe Verbreitung der Open-Source-Hardware im Hobbybereich oder auch an Hochschulen und in der IoT-Szene findet sich eine große Menge Lernmaterial und Beispielcode im Internet oder in Büchern. Da viele der Nutzer Einsteiger in Elektronik und Programmierung sind, ist die Lösung fast jedes Problems, das bei der Nutzung dieser Hardware auftreten kann, bereits im Internet zu finden. Ebenso sind gut gepflegte Softwarebibliotheken für viele Anwendungen vorhanden.

Ergebnisse

Die bei RESOL implementierten SiL- und HiL-Schnittstellen können mit einer stabilen Zykluszeit von etwa 20 ms betrieben werden. Dadurch sind der Anwendung sowohl im Entwicklungs- und Prototypingbereich als auch für Labortests und Diagnosemaßnahmen praktisch keine Grenzen gesetzt.
Durch den Einsatz dieser Techniken war es beispielsweise möglich, die Entwicklungsdauer für die Regelungssynthese eines Anlagentyps mit sehr schnellen Prozessdynamiken, welcher im Hause RESOL regelmäßig bearbeitet wird, deutlich zu reduzieren. Jede Variante dieser Anlage erfordert eine erneute Synthese. Während die Entwicklungsdauer ursprünglich etwa 160 h in Anspruch nahm, sank diese nach Einführung der besprochenen Tools und Verfahren um 75 % auf etwa 40 h. Um zwischenzeitlichen Knowhow-Zuwachs und eine Erhöhung der Handlungssicherheit im Umgang mit dem Anlagentyp zu berücksichtigen, wurde für den Vergleich die Ausgangszeit um 20 % verringert und die aktuelle Dauer um 20 % erhöht. Für einen korrekten Vergleich müsste dabei nun noch der Informationsgewinn pro Zeiteinheit berücksichtigt werden, was freilich nicht seriös durchzuführen ist. Jedoch bestätigt die Erfahrung, dass während der aktuell benötigten 40 h eine qualitativ hochwertigere und robustere Lösung erarbeitet wird als im Ausgangszustand. Dies liegt zum einen am strukturierteren, softwaregestützten Vorgehen und zum anderen an denen in der gewonnenen Zeit durchführbaren zusätzlichen Prüfungen und Weiterentwicklungen.
Im Bereich des Reglertests konnte mithilfe von HiL der Personalaufwand verringert werden. Tests die zuvor manuell durchgeführt werden mussten, laufen nun nach einmaliger Einrichtung automatisch ab. Durch zusätzlich implementierte Auswertungssoftware wird auch die Testbewertung automatisch vorgenommen.

Literatur

[1] D. Abel und A. Bollig, *Rapid Control Prototyping: Methoden und Anwendungen*. Springer Berlin Heidelberg, 2006.

[2] Y. Altman. (2017). Pause for the better. Online: `https://undocumentedmatlab.com/blog/pause-for-the-better`. Stand: 12.12.2017.

[3] J. Glembin, C. Büttner, J. Steinweg, G. Rockendorf und A. Klingenschmidt, "New control strategy for solar thermal systems with several heat sinks", 2014.